

P90X0

LIFE•WORKS  
FORMAT  
LANGUAGE  
MANUAL

*Volume I*

Hackerspace Arnhem

Broekstraat 18  
6828 PZ Arnhem  
[www.hack42.nl](http://www.hack42.nl)

#42  
HACKERSPACE ARNHEM

PHILIPS





**LifE•Works Format Language Manual,  
Volume 1**

# List of Effective Pages

This publication contains 358 pages consisting of the following:

Page Number	Date	Page Number	Date
Front Cover .....	1 Nov 87	6-1 thru 6-18 .....	1 Nov 87
Inside Front Cover (Blank) .	1 Nov 87	7-1 thru 7-53 .....	1 Nov 87
*Title .....	1 Jun 88	7-54 (Blank) .....	1 Nov 88
*A .....	1 Jun 88	8-1 thru 8-28 .....	1 Nov 87
*i .....	1 Jun 88	9-1 thru 9-3 .....	1 Nov 87
*ii (Blank) .....	1 Jun 88	9-4 (Blank) .....	1 Nov 87
iii thru vi .....	1 Nov 87	10-1 thru 10-32 .....	1 Nov 87
*vii thru ix .....	1 Jun 88	11-1 thru 11-8 .....	1 Nov 87
*x (Blank) .....	1 Jun 88	*12-1 thru 12-6 .....	1 Jun 88
1-1 thru 1-16 .....	1 Nov 87	13-1 thru 13-13 .....	1 Nov 87
2-1 thru 2-17 .....	1 Nov 87	13-14 (Blank) .....	1 Nov 87
2-18 (Blank) .....	1 Nov 87	14-1 thru 14-11 .....	1 Nov 87
3-1 thru 3-4 .....	1 Nov 87	4-12 (Blank) .....	1 Jun 88
*3-5 and 3-6 .....	1 Jun 88	A-1 thru A-26 .....	1 Nov 87
3-7 thru 3-22 .....	1 Nov 87	B-1 thru B-5 .....	1 Nov 87
*3-23 and 3-24 .....	1 Jun 88	B-6 (Blank) .....	1 Nov 87
4-1 and 4-2 .....	1 Nov 87	C-1 thru C-8 .....	1 Nov 87
*4-3 and 4-4 .....	1 Jun 88	*Index-1 .....	1 June 88
4-5 thru 4-11 .....	1 Nov 88	Index-2 thru Index-11 .....	1 June 88
4-12 (Blank) .....	1 Nov 88	*Index-12 .....	1 Jun 88
5-1 thru 5-10 .....	1 Nov 87	User's Comments .....	— — —
*5-11 and 5-12 .....	1 Jun 88	Reply Card .....	— — —
5-13 thru 5-41 .....	1 Nov 87	Inside Back Cover (Blank) ..	— — —
5-42 (Blank) .....	1 Jun 88	Back Cover .....	— — —

\*Asterisks indicate pages changed, added or deleted by the current change. The issue date for the change appears in place of the previous issue date at the bottom of each changed page included in the change package. Where a change involves a technical correction or the addition of new material, a vertical line appears at the appropriate place in the margin of the affected page. Deletions and editorial corrections are not specifically indicated.

Issue A: 15 October 1986  
Issue B: 15 February 1987  
Issue C: 1 November 1987  
Change 1: 1 June 1988

Specifications subject to change.  
Copyright ©1988  
Motorola Inc.  
All rights reserved.  
Printed in U.S.A.



## Preface

The LiFE-Works Format Language Manual, Volume 1, provides detailed information on the LiFE-Works format programming language.

This manual is intended for anyone who wants to design data entry formats under LiFE-Works. This manual describes all LiFE-Works format programming elements except those specific to Multistation File Processing (MSFP) programming.

To use this manual, you should be familiar with LiFE-Works operating procedures, data structures, and supervisory commands as outlined in the LiFE-Works Operator's Manual and the LiFE-Works Supervisor's Manual.

You may need the following manuals in addition to this manual:

- LiFE-Works Format Language Manual, Volume 2, which describes the MSFP format programming elements.
- LiFE-Works Display Messages, which describes all LiFE-Works messages.
- LiFE-Works Menu Manual, which describes the LiFE-Works menu system. The LiFE-Works menu system provides an alternate way to do many LiFE-Works procedures.

This manual applies to release VA03 of LiFE-Works which is a new product evolving from VISION II. For format programmers, LiFE-Works introduces the format generator and the data dictionary. The format generator allows you to create format code without using format language. The data dictionary prepares LiFE-Works data for use in other software applications.

This manual replaces the VISION II Format Language Manual, Volume 1. Vertical bars in the outer margin mark technical differences between the VISION II Format Language Manual, Volume 1 and this manual.

Change 1 contains a rewrite of Section 12 on interactive communications and some minor corrections.

C-ISAM is a trademark of Informix Software Inc.

INFORMIX is a trademark of Informix Software Inc.

If the documentation contained herein is supplied, directly or indirectly, to the U. S. Government, then the following notice shall apply unless agreed to in writing by Motorola Computer Systems, Inc.:

#### RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(ii) of the Rights in Technical Data and Computer Software clause at DFAR 252.227-7013.

Motorola, Inc.  
10700 North De Anza Boulevard  
Cupertino, California 95014



## Table of Contents

### 1 Introduction

- About LiFE-Works Format Programming, 1-1
- Notational Conventions Used in This Manual, 1-2
- Basic Procedures, 1-3
- Entering Format Code, 1-4
- Compiling a Format, 1-8
- Defining a Job, 1-10
- Copying a Format, 1-13
- Deleting a Format, Index Set, or Other System Element, 1-15

### 2 Field Types

- Introduction, 2-1
- Alphabetic Field Type [A], 2-3
- Alphabetic Field Type with Lowercase Feature [a], 2-4
- General Field Type [G], 2-6
- General Field Type with Lowercase Feature [g], 2-7
- Hexadecimal Field Type [H], 2-9
- Integer Field Type [I], 2-10
- Left-Zero-Fill Field Type [L], 2-11
- Left-Zero-Fill Field Type (with Implied Decimal) [Lw.d], 2-13
- Numeric Field Type [N], 2-15
- Word-Processing Field Type [W], 2-16

### 3 Field Modifiers

- Introduction, 3-1
- Auxiliary-Duplicate Modifier [A], 3-3
- Balance-Total Modifier [B], 3-6
- Check-Digit Modifier [C], 3-8
- Automatic-Duplicate Modifier [D], 3-10
- Must-Enter Modifier [E], 3-12
- Must-Fill Modifier [F], 3-13
- Generate Modifier [G], 3-14
- Index-Generate Modifier [I], 3-17
- Justify Modifier [J], 3-18
- Display-Only Modifier [Q], 3-19
- Must-Release Modifier [R], 3-20
- Skip Modifier [S], 3-21
- Execute C Function Modifier [X], 3-22
- Zero-Balance Modifier [Z], 3-24

#### 4 Attributes

Introduction, 4-1

Modifying the Appearance of an Individual Element, 4-4

Modifying the Default Display Characteristic [INTEN], 4-6

Placing an Attribute in a Specified Location [ATR], 4-8

Inhibiting Attribute Generation [-], 4-10

#### 5 Validation and Generation Descriptors

Introduction, 5-1

Accumulator Validation/Generation Descriptor [A], 5-7

Work Buffer Validation/Generation Descriptor [B], 5-8

Batch Identifier System Keyword [#BATCH], 5-9

System Constant System Keyword [#CONST], 5-10

Numeric Date System Keyword [#DATE], 5-11

Alphabetic Date System Keyword [#DATEL], 5-12

Day System Keyword [#DAY], 5-14

Field Reference Validation/Generation Descriptor [F], 5-15

Job Name System Keyword [#JOB], 5-17

Local Field Reference Validation/Generation Descriptor [LF], 5-18

Left-Justified With No Imbedded Blanks Validation Descriptor [?LJNIB], 5-20

Negative-Overpunch Character Validation Descriptor [?NEG], 5-21

Operator ID System Keyword [#OPER], 5-22

Record Number System Keyword [#RECNO], 5-23

Relative Screen Position Validation/Generation Descriptor [RS], 5-24

Screen Validation/Generation Descriptor [S], 5-26

Scan-For-Any Validation Descriptor [SA], 5-28

Scan-For-None Validation Descriptor [SN], 5-30

Terminal Number System Keyword [#TERMNO], 5-32

Time System Keyword [#TIMES], 5-33

User ID System Keyword [#USERID], 5-34

Value Set Validation Descriptor [V], 5-35

Symbolic Reference Validation/Generation Descriptor [.symbol.], 5-37

String Constant Validation/Generation Descriptor ["text"], 5-38

Hexadecimal String Constant Validation/Generation Descriptor [\], 5-41

#### 6 Screen-Formatting Commands

Introduction, 6-1

Blank to Row and Column Command [Br,c], 6-4

Blank Columns Command [Bw], 6-5

Erase to Row and Column Command [Er,c], 6-6

Erase Columns Command [Ew], 6-8

Erase Unprotected Columns to Row and Column Command [EUAr,c], 6-9

Erase Unprotected Columns Command [EUAw], 6-10

Prompt Display Command [P], 6-11

Prompt Display at Row and Column Command [Pr,c], 6-13

Generate Text Command [T], 6-15

Tab to Row and Column Command [@r,c], 6-17

Tab to Column Command [@w], 6-18



## 7 Commands

Introduction, 7-1  
Auto Skip Duplicate Command [ASD], 7-4  
Auto Skip Duplicate Off Command [ASD#], 7-5  
Audible Alarm Command [BEEP], 7-6  
Blank Work Buffer Command [BLANKB], 7-7  
Branch Command [BRANCH], 7-8  
Generate Check Digit Command [C], 7-10  
Call Subroutine Command [CALL], 7-11  
Comment Command [COMMENT or /\*comment\*/], 7-12  
Correction Command [CORR], 7-13  
Resume Character Display Command [DISPLAY], 7-14  
Inhibit Character Display Command [DISPLAY#], 7-15  
Edit Numeric Field Command [EDIT], 7-16  
End Subroutine Command [ENDSUB], 7-22  
Execute C Function Command [EXEC], 7-23  
Program Level Command [Fn], 7-25  
Force Disk Record Command [FDR], 7-26  
Get Work Buffer Command [GETBUF], 7-27  
Include Job and Batch Directive [INCLUDE], 7-28  
Lowercase On Command [LOW], 7-29  
Lowercase Off Command [LOW#], 7-30  
Enable EXIT Key Command [MODE], 7-31  
Disable EXIT Key Command [MODE#], 7-32  
No Disk Record Command [NDR], 7-33  
No Backspacing Command [NOBACK], 7-34  
No Record/Document Up Command [NOUP], 7-35  
Screen Print Command [PRINTS], 7-36  
Release Work Buffer Command [RELBUF], 7-37  
Set Command [SET], 7-38  
Nonmatching Lengths, 7-41  
Suspend Format Execution Command [SLEEP], 7-44  
Define Subroutine Command [SUBROUTINE], 7-46  
Supervisor Mode Format Commands [SUPERS and SUPERM], 7-48  
Verify On Command [V], 7-50  
Verify Off Command [V#], 7-51  
Sight Verify Command [VALID], 7-52  
Zero Accumulator Command [Z], 7-53

## 8 Arithmetic and Logic

- Introduction, 8-1
- Relational Operators, 8-1
- Arithmetic Operators, 8-3
- Logical Connectors, 8-5
- Logical OR (|), 8-6
- Logical AND (&), 8-7
- Branching Statements, 8-8
- IF...THEN...ELSE Statements, 8-9
- Nesting IF Statements, 8-12
- SPLIT Command, 8-13
- Using Conditional Indicators and Commands, 8-15
- Work Buffer Conditional Indicator [BUFFER], 8-17
- Conditional Verify Indicator [DOCVF], 8-18
- Exit Format Conditional Command [EXITF], 8-19
- Key Verify Conditional Indicator [KEYVF], 8-20
- Conditional Verify Command [KVDVF], 8-21
- Error Conditional Command [MARK], 8-22
- Error Conditional Indicator [MARKED], 8-23
- New Record Conditional Indicator [NEWREC], 8-24
- Null Conditional Command [NULL], 8-25
- Out-of-Balance Conditional Indicator [OUTBAL], 8-26
- Pass 2 Conditional Indicator [PASTWO], 8-27
- Verify Reconstruct Conditional Indicator [VRECON], 8-28

## 9 Value Sets

- Introduction, 9-1
- Creating a Value Set, 9-2
- Value Set Validation and Split Commands, 9-3

## 10 Index Sets

- Introduction, 10-1
- Creating an Index Set, 10-3
- Validating against or Generating from an Index Set, 10-5
- Exclusive Access Key Field Validation Descriptor [K], 10-8
- Shared Access Key Field Validation Descriptor [KS], 10-9
- Exclusive Access Key Field or Next Highest Key Field Validation Descriptor [KN], 10-10
- Shared Access Key Field or Next Highest Key Field Validation Descriptor [KNS], 10-11
- Index Set Record Field Validation/Generation Descriptor [X], 10-12
- Using the X Descriptor in a SET Command, 10-14
- Commands Used with Index Sets, 10-16
- Select the Previous Index Set Record Command [BACK], 10-17
- Browse Through Index Set Records Command [BROWSE], 10-19
- Delete Index Set Record Command [DELIXR], 10-21
- Insert Index Set Record Command [INSIXR], 10-22



- Select the Next Index Set Record Command [NEXT], 10-24
- Release Exclusive Access Command [R], 10-26
- Release All Access Command [RA], 10-27
- Switch Key Command [SWITCH], 10-28
- Conditional Indicators Used with Index Sets, 10-29
- Exclusive Access Conditional Indicator [EXCLUSIVE], 10-30
- Index Set Select Conditional Indicator [SELREC], 10-31
- Shared Access Conditional Indicator [SHARED], 10-32

## 11 Assigning Symbolic References

- Introduction, 11-1
- Assigning Symbols, 11-2
- Assigning Symbols in the SCREEN-SECTION, 11-5
- Multiple Symbol Assignments, 11-7
- Sharing Symbol Assignments, 11-8

## 12 Format Programming for Interactive Communications

- Introduction, 12-1
- Coding for Interactive Communications, 12-2
- Communications in Existing Records, 12-5
- Information for the Mainframe Programmer, 12-5
- Information for the Format Programmer, 12-6
  - Displaying the Attribute Character, 12-7
  - MDT Bit, 12-8
- Error Conditions, 12-10

## 13 The Data Dictionary

- Starting the Data Dictionary Generator, 13-3
- Creating a New Data Definition, 13-4
- Modifying a Data Definition, 13-10
- Deleting a Data Definition, 13-11
- Deleting a Field Definition, 13-12
- Copying a Data Definition, 13-13

## 14 LiFE-Works Format Generator

- Introduction, 14-1
- Starting the Format Generator, 14-2
- Designing a Data Input Screen, 14-4
  - Creating Screen Prompts, 14-6
  - Creating Data Fields, 14-6
  - Defining Data Fields, 14-6
    - Field Type Menu, 14-7
    - Field Attributes Menu, 14-8
    - Field Characteristics Menu, 14-8
    - Field Validation Menu, 14-9
    - Field Processing Menu, 14-10
- Compiling the Code, 14-11

#### 14 LiFE-Works Format Generator

- Introduction, 14-1
- Starting the Format Generator, 14-2
- Designing a Data Input Screen, 14-4
  - Creating Screen Prompts, 14-6
  - Creating Data Fields, 14-6
  - Defining Data Fields, 14-6
    - Field Type Menu, 14-7
    - Field Attributes Menu, 14-8
    - Field Characteristics Menu, 14-8
    - Field Validation Menu, 14-9
    - Field Processing Menu, 14-10
- Compiling the Code, 14-11

#### A Record Processing

- Introduction, A-1
  - Pass 1 (Display), A-1
  - Pass 2 (Execution), A-2
  - Pass 3 (Update), A-3
- Using the PASTWO Conditional Indicator, A-3
- Application Examples, A-5
  - Example 1: Batch Balancing, A-5
  - Example 2: Index Set Update, A-7
  - Implementation of Pass 1, Pass 2, and Pass 3, A-7
  - Relationship Between Pass 1, Pass 2, and Pass 3, A-9
  - Effect on Batch Balancing, A-12
  - Effect on Index Set Update, A-17
  - A Sample Problem, A-21
  - Discussion of Sample Problem, A-22
  - Other Problems, A-24

#### B Check Digit Calculation

- Introduction, B-1
- Modulus 7, B-2
- Modulus 10, B-3
- Modulus 11, B-4
- Alphabetic Check Digits, B-5

#### C The LiFE-Works Editor

- Introduction, C-1
- The Mode M Search Function, C-6





## Illustrations

- 13-1 The Data Dictionary Generator Menu, 13-3
- 13-2 The Data Definition Menu, 13-4
- 13-3 The Add New Field Definitions Menu, 13-6
- 13-4 The Fields Defined for Data Definition name Menu, 13-7
- 14-1 Format Generator Menu, 14-2
- 14-2 Status Line, 14-4
- 14-3 Field Definition Menu, 14-7
- 14-4 Field Type Menu, 14-7
- 14-5 Field Attributes Menu, 14-8
- 14-6 Field Characteristics Menu, 14-9
- 14-7 Field Validation Menu, 14-9
- 14-8 Field Processing Menu, 14-10
- 14-9 Source Code Already Exists Menu, 14-11
- A-1 Sample Screen Display, A-4
- A-2 PASTWO Format Code, A-4
- A-3 Creating a New Record, A-10
- A-4 Modifying an Existing Record, A-11

## Tables

- 2-1 Field Types, 2-1
- 3-1 Field Modifiers, 3-1
- 4-1 Commands Related to Attributes, 4-2
- 5-1 Validation/Generation Descriptors, 5-2
- 6-1 Screen-Formatting Commands, 6-1
- 7-1 Commands, 7-2
- 7-2 EDIT Picture Symbols, 7-18
- 8-1 Conditional Indicators and Commands, 8-16
- 10-1 Validation/Generation Descriptors Used with Index Sets, 10-5
- 10-2 Commands Used with Index Sets, 10-16
- 10-3 Conditional Indicators Used with Index Sets, 10-29
- 12-1 Format Programming Elements Used in Interactive Communications, 12-2
- 12-2 Bit Settings in the Attribute Byte, 12-7
- 12-3 Display of Attribute Bytes, 12-8
- 13-1 Keys Used in Data Dictionary Menus, 13-1
- 14-1 Format Generator Function Keys, 14-5
- A-1 Combinations of NEWREC and PASTWO, A-17
- C-1 Mode M Function Keys, C-3
- C-2 Function Keys for Mode M Searches, C-6
- C-3 Operating System Metacharacters Used in Mode M Searches, C-8







## Section 1 Introduction

### ABOUT LIFE-Works FORMAT PROGRAMMING

The LIFE-Works format programming language lets you design data entry screens and manipulate entered data. For example, a simple format program might:

1. Display a screen prompt to the operator for each item of information to be entered.
2. Provide a space in which the operator can enter the requested information.
3. Process the entered data. For example, the format might check an entered name against a list, or add two numeric fields together and display the resulting total.
4. Store the data in a file.

As a format programmer, you can create programs that perform simple data collection or complex data processing. Here are some of the things a format program can do:

- Create varied screen displays. For example, you can display a prompt in high intensity or make an error message flash.
- Place restrictions or qualifications on entry fields. For example, you can specify that the operator must fill a given field completely.
- Check the operator's entry against a list of valid entries.
- Copy data from another source into the current record.
- Test for certain conditions and do different kinds of processing based on the results of the test.
- Maintain totals and keep running counts.
- Perform arithmetic operations.
- Run subroutines or execute batches of supervisory commands.

NOTATIONAL CONVENTIONS USED IN THIS MANUAL

This manual uses the following notational conventions in command strings and examples:

UPPERCASE      Uppercase letters represent format code elements, supervisory commands, and other literals. For example, \$FORMAT is a command that must be spelled exactly as shown. You can use upper- or lowercase letters unless otherwise specified.

lowercase      Lowercase letters represent variables. For example, job represents a job name.

[]              Brackets indicate that the enclosed element is optional. For example, [\$PASSW=password] indicates that a password is optional.

{ }             Braces indicate alternate choices. For example,

$$\left\{ \begin{array}{l} \$SIZE \ n \\ \$MAXSIZE \ n \end{array} \right\}$$

indicates that you must use either \$SIZE or \$MAXSIZE.

...             Ellipses indicate that the preceding element can be repeated. For example, \$VALUES id[,id...] indicates that you can supply a series of value-set identifiers, separated by commas.



## BASIC PROCEDURES

The basic steps in creating and using a format program are:

1. Enter the format code. A format program consists of a series of commands entered into general-format records, much as supervisory commands are entered. For example:

```
$FORMAT 001
```

```
E1760 P"NAME: " A25
```

```
B2,1 P"ADDRESS: " G30
```

2. Compile the format code. Compiling translates the entered format commands into a stored system element that can be used by any number of jobs.
3. If necessary, enter and compile any other system elements required by the format. These might include value sets, index sets, MSFP forms, or MSFP Ftypes.
4. Create a job definition. The job definition assigns one or more formats to a job. Each format named in the job definition becomes a program level of the job.

Procedures for entering and compiling format code and for creating a job definition (steps 1, 2, and 4 above) appear in this section. For information on entering and compiling other system elements (step 3 above), refer to the following:

- Section 9, "Value Sets," for information on value sets.
- Section 10, "Index Sets," for information on index sets.
- The LiFE-Works Format Language Manual, Volume 2 for information on MSFP Ftypes and forms.

### NOTE

The LiFE-Works menu system is an alternate way to do many LiFE-Works operations. For example, you can define a job or create an index set by using the menu system. For full information on the LiFE-Works menus, see the LiFE-Works Menu Manual.

## ENTERING FORMAT CODE

You can enter format code into a batch of MASTER or into any job defined with general-format records and no screen prompts. The procedure is similar to entering supervisory commands.

### Procedure

- 1 To start a format program, enter the following command:

`$FORMAT id [subcommands]`

where

id is a unique 3-character identifier to be assigned to the format. Any combination of the numbers 000 to 999 and the letters AAA to FFF is valid.

subcommands are any of the following subcommands:

\$ATTRSPACE YES or \$ATTRSPACE NO to override your system's attribute configuration. If your system is configured for attribute spaces but you don't want attribute spaces in this format, include the \$ATTRSPACE NO subcommand. If your system isn't configured for attribute spaces but you want attribute spaces in this format, include the \$ATTRSPACE YES subcommand.

\$DOCHDR to assign this format as a document header.

\$DOUBLE to indicate that the format program uses double-precision accumulators (24 characters long). If you don't use \$DOUBLE, single-precision accumulators (12 characters) are assumed.

\$MAXCOL n to specify the maximum line length. n must be either 80 or 96. If you don't include \$MAXCOL, the system uses the line length set during configuration.

\$MAXROW n to set a limit on the number of screen rows this format can use. n can be a number from 1 to 24. If you don't include \$MAXROW, the format can use all 24 screen rows, including the message line and status line.

\$SCREEN s to set a limit on the number of characters the format can place on the screen. s can be any number from 1 to 1920. If you don't include \$SCREEN, the format can use the entire screen, including the message line and status line.

\$SIZE n to document the number of characters in the data record this format creates. n can be a number from 1 to 1500.

\$SUB is required if this format uses subroutines.

\$SYM is required if this format uses symbolic references.

- 2 On the next record after the \$FORMAT command, start entering the format code. Depending on what you want to accomplish, this can be a fairly brief process or a rather lengthy one. Most of the rest of this manual is devoted to describing the LIFE-Works format code elements.

#### Usage Notes

- The entire \$FORMAT command (including any subcommands) must appear in a record by itself and must be contained in a single record. If necessary, you can use the backslash continuation character (CTRL /) to continue the \$FORMAT command to the next record.
- Check the format directory (mode S-F) to make sure the format identifier you choose is not already in use. The identifier 999 is reserved; format 999 always exists on your system and cannot be changed or deleted.
- The syntax rules for entering format code are:
  - You can use uppercase and lowercase letters in any combination unless otherwise specified.
  - You must put at least one separating blank between individual format code elements. You can use additional blanks as desired to make the code more readable. For example:

B3,15      P"DEPT. NO.:"      T"5437"

- You don't have to break lines of code in specific places. It is possible--though not recommended--to enter code in a continuous stream, wrapping from record to record. Your code will be easier to read and debug if you put one or two elements, or a related group of elements, in each record. For example:

```
IF F1>"50"
```

```
THEN SET S(10,20,9)="TOO LARGE"
```

```
ELSE NULL
```



## Section 1

### Introduction

The continuation character (\) is not needed or recognized within format code. (You can use the continuation character in supervisory commands associated with format code, such as the \$FORMAT command.)

- Some format code elements, such as character strings, must be enclosed in quotation marks. You can use either double or single quotation marks as long as you use matched pairs. In other words, if the beginning quotation mark is single, the ending quotation mark must also be single.

You can also use quotation marks within a character string. For example:

"'Alas,' cried Pearl"

or

'"Alas," cried Pearl'

- LiFE-Works uses a few nontraditional symbols, as follows:

<u>Meaning</u>	<u>Traditional symbol</u>	<u>LiFE-Works symbol</u>
not equal	$\neq$	! (exclamation point)
general divide	$\div$	% (percent)

For example:

IF F3!'100'... means "if field 3 is not equal to 100..."

SET F4=(F2+F3)%2 means "set field 4 equal to the value produced by adding field 2 to field 3 and then dividing the total by 2."

- To make debugging easier, enter a \$PAUSE command and a \$DELETE command at the beginning of each format. For example:

```
$PAUSE
$DELETE FORMAT 222
$FORMAT 222
```

The \$PAUSE command makes the system pause before each format. This is useful when you have several formats in a single batch; you can compile or recompile each format individually. The \$DELETE command automatically deletes the old object code each time you recompile a format.

- A multirecord editor, which lets you view and edit a screenful of records, is a useful tool for entering and editing format code. You can access two different multirecord editors from LiFE-Works.

Select mode M to access a multirecord editor that is part of LiFE-Works. See Appendix C, "The Mode M Editor," for instructions on using this editor.

Select mode I to access the operating-system editor vi. See the System V/68 User's Guide for instructions on using vi. When used on LiFE-Works batches, vi has the following limitations:

- It can be used only on fixed-length batches. It can handle any record length, but all record lengths must be the same for the batch.
- It is best not to use vi on batches with multiple program levels. If used to edit batches that have more than one program level, vi changes all edited material to program level 1.

### Examples

```
$FORMAT 123
```

Creates format 123.

```
$format 888 $size 80 $dochdr
```

Creates format 888 and assigns it as a document header. The \$SIZE subcommand documents the size of the record this format creates (80 characters).

```
$Format 001 $Double $Sym
```

Creates format 001, which uses double-precision accumulators and symbolic referencing.

### COMPILING A FORMAT

Format code elements as you originally enter them into a job and batch are known as source code. To make the source code into a usable system element, you must compile it. Compiling transforms the source code into object code, a language that the system understands.

When you compile format source code, LiFE-Works creates a separate object code file and assigns it the three-digit identifier you specified in the \$FORMAT command. This object code file actually performs the operations described in the source code--producing screen prompts, manipulating data, and so on.

### Procedure

To compile a format, execute the batch that contains the format code as you would execute any batch of supervisory commands.

- If you are currently working in the batch in mode E or mode F, display the first line of the format program, then select mode T-M.
- If you are working in the batch in mode M or if you are in some other mode, select mode T-M. LiFE-Works prompts you to enter a job and batch name. If the status line contains the name of the job and batch you want to execute, press DUP; if not, type the desired job and batch name, then press RETURN.

In either case, the messages

Format successfully compiled

and

End-of-Batch reached in M-option

signal that the format has compiled.

### Usage Notes

- Compiling does not affect the source code file. You can access the source code file and edit it as desired. However, the changes have no effect on the format until you delete the compiled object code and recompile the format. See "Deleting a Format, Index Set, or Other System Element" in this section for information on deleting object code.
- If the code contains an error that prevents it from compiling, an error message appears on the message line and the line of code containing the error appears at the top of the screen. The cursor is usually located on or near the error. Press CORR to move the cursor into the code, and make whatever changes are necessary.



If you need further explanation of an error message, see the LIfE-Works Message Manual.

- Formats cannot be used until they are assigned to a job by a job definition statement. (See "Defining a Job" in this section.)

## DEFINING A JOB

Once you have compiled your format code and any system elements the format refers to (value sets, index sets, and so on), you are ready to define a job that uses the format.

### Procedure

To define a job, enter and execute the command

$\$DEFJOB \text{ job } \left\{ \begin{array}{l} \$SIZE \text{ } n \\ \$MAXSIZE \text{ } n \end{array} \right\} \$FORMAT \text{ id[,id...]} [\text{subcommands}]$
--

where

job is the name to be assigned to the job. The rules for job names are:

- Job names can be one to nine characters long.
- Job names can include upper- and lowercase letters, numbers, and all symbols except slashes (/), backslashes (\), and leading periods. If you use embedded blanks, commas, or semicolons, you must enclose the job name in quotation marks. See "Usage Notes" below for more information.
- Alphabetic case is significant; MASTER, Master, and master are three different job names.

n specifies the size in characters (including spaces) of the records used in this job. The maximum size of the largest record can be 1500 characters. Use \$SIZE for fixed-length records; use \$MAXSIZE for variable-length records.

id is the three-character identifier of each format to be used in the job. You can repeat id up to 15 times, separating the identifiers with commas.

subcommands are any of the following subcommands:

\$ACCUMS n[D] to assign n accumulators to the job. n can be any number between 1 and 23 (single-precision accumulators) or between 1 and 12 (double-precision accumulators, indicated by D). If you don't use the \$ACCUMS subcommand, five accumulators are automatically assigned to the job.

\$PASSWORD password to assign the password password to this job. password can contain from one to eight characters; LIFE-Works adds trailing blanks as necessary to create an eight-character password.

\$VALUES id[id...] to assign the value sets this job uses. id is the three-character value set identifier. You can repeat id up to 15 times, separating the identifiers with commas.

### Usage Notes

- The \$DEFJOB command statement must appear in a single record. If necessary, you can use the backslash continuation character (CTRL /) to continue the command on the next record.
- Using blanks, commas, or semicolons in a job name imposes some significant limits on the job. Such a job cannot be accessed in modes E, F, V, M, or I; any changes must be done through format code or supervisory commands. You must enclose the name in quotation marks in the \$DEFJOB command and in any format command or supervisory command that refers to the job. (If a given command already requires that the job name appear in quotation marks, you need not add a second set of quotation marks.)
- The order in which you list formats and value sets in the \$DEFJOB command is important. The first format listed after the \$FORMAT subcommand becomes program level 1 for that job, the second format listed becomes program level 2, and so on. Similarly, the first value set listed after \$VALUES becomes value set 1 for the job.

The same formats and value sets can be assigned to more than one program level within a job and to more than one job.

You can add blanks between format or value set identifiers, but separating commas are required even if you use blanks.

- You can use the alternate form \$MAXSIZ instead of \$MAXSIZE, and the alternate form \$PASSW instead of \$PASSWORD.
- When a job does not require accumulators, specify \$ACCUNS 0 to save disk space.

If you specify double-precision accumulators in the \$DEFJOB command, all formats for the job must have been defined with the \$DOUBLE subcommand in the \$FORMAT statement.

- If you set a password on a job, the password must be entered each time the job is accessed. For increased security, you can blank the password in the \$DEFJOB command after the job is defined.
- You can create a job definition that refers to nonexistent formats or value sets. However, you cannot use the job until all system elements it requires have been entered and compiled.
- You can redefine an existing job as long as you purge any data in the job first. Make sure you have an adequate backup of any data you need to save.



Section 1  
Introduction

Examples

```
$DEFJOB PAYROLL $MAXSIZ 120 $FORMAT 800,803,805
```

Defines the job PAYROLL, a variable-length job with three program levels (\$FORMAT 800,803,805) and a maximum record size of 120 characters (\$MAXSIZ 120).

```
$Defjob Newjob $Size 80 $Format 912 $Values 303,304 $Passw secret
```

Defines the job Newjob, a fixed-length job with one program level (\$Format 912) and 80-character records (\$Size 80). Newjob uses two value sets and is password-protected.

```
$defjob ndebt $size 100 $format 222 $accums 4d
```

Defines the job ndebt, a fixed-length job with one program level (\$format 222) and 100-character records (\$size 100). ndebt uses four double-precision accumulators.

## COPYING A FORMAT

You can copy a compiled format (an object-code file) to another LiFE-Works file or to an operating system file. You can also copy a compiled format back from the operating system into LiFE-Works.

### Procedure

To copy a compiled format, enter and execute:

```
$COPY { $FORMAT id } { TO } { $FORMAT id }  
      { $PATH file } { OVER } { $PATH file }
```

where

id is the three-character format identifier. On the source side, id is the format to be copied. On the destination side, id is the format identifier to which the format is to be copied.

file is the full pathname of the operating system file. On the source side, file is the file to be copied. On the destination side, file is the file into which the format is to be copied.

### Usage Notes

- If either the source or destination file resides on another OfficeLAN node, specify the node immediately before the source or destination file name. (See "Examples" below.) You can copy a file from one node to another even if you are on a third node.
- For general information about the \$COPY command, see the LiFE-Works Supervisor's Manual.

### Examples

```
$COPY FORMAT 100 TO /tmp/100
```

Copies object code for format 100 to the operating system file /tmp/100.

Section 1  
Introduction

\$COPY \$FORMAT 123 TO \$NODE MKTG \$FORMAT 222

Copies object code for format 123 into format 222 on the OfficeLAN  
node MKTG.



### DELETING A FORMAT, INDEX SET, OR OTHER SYSTEM ELEMENT

System elements such as formats and index sets can be used by many jobs. Be careful not to delete an element that is in use.

#### Procedure

To delete object code for jobs, formats, value sets, index sets, MSFP Ftypes, or MSFP forms, enter and execute the following supervisory command:

\$DELETE	{	JOB job	}
		FORMAT id	
		VALUES id	
		INDSET id	
		Ftype id	
		FORMS id	

where

job is the name of the job to be deleted.

id is the identifier of the format, value set, index set, MSFP Ftype, or MSFP form to be deleted.

The message REQUEST COMPLETE appears if the deletion was successful; otherwise an error message appears.

#### Usage Notes

- See the Life-Works Supervisor's Manual for more information on deleting jobs.
- When you delete an index set, the index set identifier is deleted from the index set directory and the ISAM flag is cleared from the batch originally used to create the index set. Any data in the index set is written back into the original batch. To eliminate the data, clear the index set by executing a \$CLEARIX command before deleting the index set. See the Life-Works Supervisor's Manual for information on \$CLEARIX.

#### Examples

\$DELETE FORMAT 123
---------------------

Deletes format 123.

\$CLEARIX 123

\$DELETE INDSET 123

Clears all data from index set 123, then deletes the index set.







Section 2  
Field Types

INTRODUCTION

A field type defines the type of data that can be entered in a field. For example, you can create a field that accepts only letters and spaces, or a field that accepts only numbers. Table 2-1 describes the LiFE-Works field types.

Table 2-1. Field Types

Field Type	Description
A	Alphabetic (A-Z or spaces)
a	Alphabetic (A-Z or spaces with lowercase feature)
G	General (any entry)
g	General (any entry with lowercase feature)
H	Hexadecimal (0-9, A-F)
I	Integer (0-9) only
L	Left-zero-fill, integer (0-9 only)
Lw.d	Left-zero-fill with implied decimal point, integer (0-9 only)
N	Numeric (any characters except A-Z)
W	Word-processing (any characters, with word wrap and text-editing features)

General Usage Notes

- Each field type must include the length of the field. For example, A25 defines an alphabetic field 25 characters in length.
- The maximum length of a field is 1500 characters.

## Section 2

### Field Types

- Any field type except W can be prefixed with an identifying field number. The field number lets you refer to the field in another format code statement. For example, the statement 1A25 defines a 25-character alphabetic field and assigns it as field 1. You can then refer to field 1 elsewhere in the format code.

A field number can be any whole number from 1 to 999. You can assign up to 999 field numbers in a given format program.

You cannot use the same field number more than once in a format statement. Certain format statements, such as IF and SPLIT statements, require that you refer to the same field in each branch of the statement; in this case, the field number is implied after the first reference. See Section 8, "Arithmetic and Logic," for details.

- All data entry fields are written to the disk unless you specify otherwise.
- As the operator enters data, all fields except the G, g, and W fields are checked, character by character, to ensure that the data matches the field type specified. Validation checks and modifiers attached to the field execute when the operator fills the entire field or releases the field by pressing RETURN. The operator can override a validation check by pressing the VALID key.

Key verify mode compares each field against the previous operator's entry, not against the field type. The key verify operator can enter data that does not match the field type as long as it matches the previous operator's entry. Key verify mode does check the field type if the operator opens the record for correction.

- You can alter the appearance of display prompts and data entry fields by using attribute modifiers. (See Section 4, "Attributes.")
- In the syntax description for each command in this section, the literal part of the command (the part you type exactly as shown) is underlined for clarity. For example, the syntax for the A field type is shown as:

[f] A[(i)]w



### ALPHABETIC FIELD TYPE [A]

Use the A field type to specify a field that accepts only the letters A through Z or spaces, and that forces letters to uppercase.

The syntax for the A field type is:

[f]A[(i)]w

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the width (length) of the field in columns.

### Usage Notes

- You must use an uppercase A to indicate the A field type.
- During data entry, the A field type works like the shift key on a typewriter. It forces all lowercase entries to uppercase, so all entries for the A field appear as uppercase on the data entry screen and in the data record.
- The A field type can be used with any field modifier and any validation/generation descriptor.
- See also "Alphabetic Field Type with Lowercase Feature [a]" in this section.

### Examples

A25

A 25-character alphabetic field.

2A(F)25

A 25-character alphabetic field that includes an optional field number (2) and an optional attribute modifier for a flashing field (F).

Section 2  
Field Types

ALPHABETIC FIELD TYPE WITH LOWERCASE FEATURE [a]

Use the a field type to specify a field that accepts only the letters A through Z or spaces, and that displays letters in both upper- and lowercase.

The syntax for the a field type is:

[f]a[(i)]w

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the width (length) of the field in columns.

Usage Notes

- You must use a lowercase a to indicate the a field type.
- Use the a field type with the lowercase feature on (LOW) so the field can have both upper- and lowercase entries. (The lowercase feature is on at the start of a format.) If you turn the lowercase feature off (LOW#), all entries are forced to uppercase.

The lowercase command can appear anywhere in the format program as long as it appears before the desired field. The lowercase feature remains on until you turn it off, and remains off until you turn it on.

- The a field type can be used with any field modifier and any validation/generation descriptor.
- See also "Alphabetic Field Type [A]" in this section.

Examples

a25

A 25-character alphabetic field.

2a(F)25

A 25-character alphabetic field that includes an optional field number (2) and an optional attribute modifier for a flashing field (F).



### GENERAL FIELD TYPE [G]

Use the G field type to specify a field that accepts any character from the keyboard, and that forces letters to uppercase.

The syntax for the G field type is:

[f]G[(i)]w

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the width (length) of the field in columns.

### Usage Notes

- You must use an uppercase G to indicate the G field type.
- During data entry, the G field type works like a shift key on a typewriter. It forces all lowercase entries to uppercase entries. Numbers and characters remain unshifted.
- The G field type can be used with any field modifier and with any validation/generation descriptor.
- See also "General Field Type with Lowercase Feature [g]" in this section.

### Examples

G9

A nine-character general field.

1G(U)9

A nine-character general field that includes an optional field number (1) and an optional attribute modifier for an underscored field (U).

GENERAL FIELD TYPE WITH LOWERCASE FEATURE [g]

Use the g field type to specify a field that accepts any character from the keyboard, and that displays letters in both upper- and lowercase.

The syntax for the g field type is:

[f]g[(i)]w

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the width (length) of the field in columns.

Usage Notes

- You must use a lowercase g to indicate the g field type.
- Use the g field type with the lowercase feature on (LOW) so the field can have both upper- and lowercase entries. (The lowercase feature is on at the start of a format.) If you turn the lowercase feature off (LOW#), all letters are forced to uppercase.

The LOW and LOW# commands can appear anywhere in the format program as long as they appear before the desired field. The lowercase feature remains on until you turn it off and off until you turn it on. (See Section 7, "Other Commands," for a detailed description of the LOW and LOW# commands.)

- The g field type can be used with any field modifier and with any validation/generation descriptor.
- See also "General Field Type [G]" in this section.

Examples

g9

A nine-character general field.

Section 2  
Field Types

1g(U)9

A nine-character general field that includes an optional field number (1) and an optional attribute modifier for an underscored field (U).



### HEXADECIMAL FIELD TYPE [H]

Use the H field type to specify a field that accepts any hexadecimal character (0-9, A-F). The H field type is not commonly used for data entry. However, it is useful for certain programming functions, such as communicating with a mainframe.

The syntax for the H field type is:

[f]H[(i)]w

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the width (length) of the field in columns.

### Usage Notes

- The H field type can be used with any field modifier and with any validation/generation descriptor.

### Examples

H10

A ten-character hexadecimal field.

3H(L)10

A ten-character hexadecimal field that includes an optional field number (3) and a optional attribute modifier (L) for a low-intensity field display.

Section 2  
Field Types

INTEGER FIELD TYPE [I]

Use the I field type to specify a field that accepts only integers (0-9).

The syntax for the I field is:

[f]I[(i)]w

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the width (length) of the field in columns.

Usage Notes

- Use the I field type for numbers that don't represent values, such as zip codes.
- The I field does not accept decimal points or negative numbers.
- The I field type can be used with any field modifier and with any validation/generation descriptor.

Examples

I7

A seven-character integer field.

4I(H)7

A seven-character integer field that includes an optional field number (4), and an optional attribute modifier for high-intensity field display (H).

LEFT-ZERO-FILL FIELD TYPE [L]

Use the L field type to specify a right-justified zero-filled field that accepts only integers (0-9).

The syntax for the L field is:

[f]L[(i)]w

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the width (length) of the field in columns.

Usage Notes

- When the operator releases an L field, the data is right-justified and any empty columns on the left are filled with zeroes.

The operator can release an empty L field. If the operator presses RETURN in the first column of an empty L field, the entire field is blank-filled. If the operator presses SHIFT RETURN in the first column of an empty L field, the entire field is zero-filled.

- You can use the L field for negative values. For negative values, the operator must release the field with the minus (-) key. The format program replaces the rightmost digit in the field with a negative overpunch character. (The term "negative overpunch" is borrowed from punch cards.)

A negative overpunch character represents a single-digit value (0 through 9) and signifies that the value is negative. The negative overpunch characters are:

<u>Number</u>	<u>Negative Overpunch Character</u>
0	}
1	J
2	K
3	L
4	M
5	N
6	O
7	P
8	Q
9	R



Section 2  
Field Types

- The L field type can be used with any field modifier except the must-release (R) modifier; a left-zero-fill field is automatically a must-release field. The must-fill modifier (F) is legal to use, but it is ignored. L can be used with any validation/generation descriptor.
- See also "Left-Zero-Fill Field Type (with Implied Decimal) [Lw.d]" in this section.

Examples

L5

A five-character left-zero-fill field.

2L(L)5

A five-character left-zero-fill field that includes an optional field number (2) and an optional attribute modifier for a low-intensity field display (L).

LEFT-ZERO-FILL FIELD TYPE (WITH IMPLIED DECIMAL) [Lw.d]

Use the Lw.d field type to specify a right-justified, zero-filled field that accepts only integers (0-9) and that has an implied decimal point at a specified position. This field type is commonly used for dollar amounts.

The syntax for the Lw.d field is:

[f]L[(i)]w.d

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the width (length) of the field in columns. The decimal point does not occupy a space and is not included in the total width of the field.

d is the number of places to the right of the decimal point. d can be any number from 1 to 15.

Usage Notes

- The Lw.d field accepts a decimal point, but the decimal point does not appear on the screen or in the data record.
- You can't specify an Lw.d field that accepts a value smaller than 1. In other words, the total width of the field (w) must exceed the number of specified decimal places (d) by at least 1.
- When the operator releases an Lw.d field, the field is right-zero-filled to the specified number of decimal places, right-justified, then left-zero-filled to the specified width of the field.

The operator can release an empty Lw.d field. If the operator presses RETURN in the first column of an empty Lw.d field, the entire field is blank-filled. If the operator presses SHIFT RETURN in the first column of an empty Lw.d field, the entire field is zero-filled.

- You can use an Lw.d field for negative values. For negative values, the operator must release the field with the minus (-) key. The format program replaces the rightmost digit in the field with a negative overpunch character. (The term "negative overpunch" is borrowed from punch cards.)

A negative overpunch character represents a single-digit value (0 through 9) and signifies that the value is negative. The negative overpunch characters are:

Section 2  
Field Types

<u>Number</u>	<u>Negative Overpunch Character</u>
0	}
1	J
2	K
3	L
4	M
5	N
6	O
7	P
8	Q
9	R

- The Lw.d field type can be used with any field modifier except the must-release (R) modifier; a left-zero-fill field is automatically a must-release field. The must-fill modifier (F) is legal to use, but it is ignored. Lw.d can be used with any validation/generation descriptor.
- See also "Left-Zero-Fill Field Type [L]" in this section.

Examples

L5.2

A five-character left-zero-fill field with two decimal places. If a data item of 10.5 is entered for this field, the value 01050 is displayed on the screen.

2L(L)5.2

A five-character left-zero-fill field with two decimal places. The field description includes an optional field number (2) and an optional attribute modifier for a low intensity field display (L).



### NUMERIC FIELD TYPE [N]

Use the N field type to specify a field that accepts numbers and all other characters except alphabetic characters.

The syntax for the N field type is:

[f]N[(i)]w

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the width of the field in columns.

#### Usage Notes

- The N field type accepts negative numbers, but they cannot be used for calculation. To enter a negative number, the data entry operator presses the minus key before typing the number.
- The N field type can be used with any field modifier and with any validation/generation descriptor.

#### Examples

N7

A seven-character numeric field.

2N(F)7

A seven-character numeric field that contains an optional field number (2) and an optional attribute modifier (F) for a flashing display.

### WORD-PROCESSING FIELD TYPE [W]

Use the W field type to specify a field that accepts any characters and that includes text entry features such as word wrap and tab settings. A W field is useful when large amounts of text must be entered.

The syntax for the W field type is:

W[(i)]w,h

where

i is an optional attribute modifier. i must be enclosed in parentheses.

h is the height of the field in rows (1-24).

w is the width (length) of the field in columns (1-80).

### Usage Notes

- You can think of a W field as a rectangular, borderless "box" on the screen. The maximum size of the entire field (height times width) is 1500 characters. The "box" must fit on the screen; for example, you cannot define a 25-column field that starts in column 75.

For programming purposes, a W field takes up all screen positions from its upper left corner through its lower right corner. In other words, even though the W field itself may not use full rows, you cannot place any other elements beside the W field. (You can place elements immediately before or after a W field, but no element can extend into a W field.)

- The W field does not change alphabetic case. If the lowercase feature is on, the W field displays both upper- and lowercase letters as entered; if the lowercase feature is off, the W field displays only uppercase letters.
- Only the D and S field modifiers can be used with a W field.
- You cannot assign a field number to a W field. Therefore, you cannot refer to a W field in a conditional statement or generate text into a W field by using the G or A modifiers.
- You can use a SET command to move data into or out of a W field, but you must move one line at a time.
- A W field cannot be key verified. If verify is on when the key verify operator reaches a W field, LIfE-Works turns verify off (V#) for the field, then turns verify on (V) at the end of the field.

- If your system uses attribute spaces, be aware that each row of a W field begins and ends with an attribute. For example, a W field defined as 20 characters wide actually occupies 22 columns on the screen—one column for the beginning attribute, 20 columns for the field itself, and one column for the ending attribute.
- A W field must have an attribute; you cannot suppress the attribute by using a minus sign (-).
- See the LIfE-Works Operator's Manual for details of cursor movement and text editing in a W field.

Examples

W5,70

A word-processing field that is five rows high and 70 columns wide.

W(B1,C3)10,40

A word-processing field that is ten rows high and 40 columns wide, with background color 1 and character color 3 (B1,C3).



# THE UNIVERSITY OF CHICAGO

OFFICE OF THE DEAN OF STUDENTS  
540 EAST 58TH STREET, CHICAGO, ILL. 60637  
TELEPHONE 733-4331

For information regarding admission, financial aid, and other matters, please contact the Office of the Dean of Students. We are pleased to assist you in your academic journey.

Yours faithfully,  
[Signature]  
Dean of Students







Section 3  
Field Modifiers

INTRODUCTION

Field modifiers provide additional information about how data is entered into a field and how it is handled by the system. Table 3-1 lists the field modifiers.

Table 3-1. Field Modifiers

Modifier	Description
A	Generates data into a field when the operator presses AUX DUP. The data can be generated from a character string, a numeric constant, an arithmetic expression, or a validation/generation descriptor.
B	Adds the contents of a field to an accumulator.
C	Checks the check digit in the rightmost position of a number.
D	When the auto skip duplicate feature (ASD) is on, duplicates data left on the screen from the previous record into the same field in the current record.
E	Must enter this field.
F	Must fill this field.
G	When the auto skip duplicate feature (ASD) is on, automatically generates fields from character strings, numeric constants, arithmetic expressions, and validation/generation descriptors.
I	Generates a field from a value set when the operator presses the INDEX key.
J	Right-justifies this field with blank fill.
Q	Display this field only (do not write to disk).
R	Must release this field.
S	Skips automatically and blanks skipped columns if ASD is turned on.
X	Passes this field value to a specified C function, and executes the function.
Z	Subtracts from an accumulator.

### Section 3 Field Modifiers

#### General Usage Notes

- Field modifiers can be used only with the field types described in Section 2. You cannot use them to modify any other format programming element.
- You can use multiple modifiers on a field as long as the modifiers are a logical combination. For example, A4ER specifies a four-character alphabetic field (A4) that is both a must-enter field (E) and a must-release field (R).

You can place separating periods between multiple modifiers for readability (A4E.R). You cannot put spaces between the field type and the modifiers, nor between two modifiers.

- For clarity, the literal part of each modifier (the part you type exactly as shown) is underlined in the syntax description. For example, the syntax for the A modifier is shown as:

fieldAd

### AUXILIARY-DUPLICATE MODIFIER [A]

Use the A modifier to generate data into a field when the data entry operator presses the AUX DUP key.

The syntax for the A modifier is:

fieldAvalue

where

field is the field specification (type, length, optional field number, optional attribute modifier).

value is either the value to be generated or a source from which the value is to be generated. value can be a character string, a numeric constant, an arithmetic expression, a system keyword, or one of the following validation/generation descriptors:

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c for a length of l characters.

Ff refers to field f of the current format.

LFf refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c for a length of l characters.

RS(n,l) refers to a relative screen position, beginning n columns from the initial cursor position for a length of l characters.

S(r,c,l) refers to the screen area beginning in row r, column c, for a length of l characters.

X(id,c,l) refers to index set id, beginning in column c for a length of l characters.

.symbol. refers to a name previously assigned to a screen area, or a portion of an index set record, work buffer, or MSFP record buffer.

### Usage Notes

- LIFE-Works uses accumulator 0 as a temporary storage area for numeric constants or arithmetic expressions. Therefore, if you are generating from a numeric constant or arithmetic expression, the length of the field cannot exceed the accumulator length (12 columns for single-precision accumulators, 24 columns for double-precision accumulators).



Section 3  
Field Modifiers

- If the generated value is a character string:
  - The entire string must be enclosed in quotation marks.
  - The string cannot exceed 180 characters.
  - The number of characters in the string must be less than or equal to the number of characters specified in the field type. The format program fills any unused spaces with blanks to the right of the last character.
- A generated field overrides the specified field type. For example, if you specify that a string of digits is to be generated into an alphabetic field, the format program will do so, without producing any error message.
- You can use the A modifier with all field types except W and with all other field modifiers except I and G.
- If the length of the generated value does not match the length of the field, LiFE-Works truncates or pads the data. See the discussion of nonmatching lengths in Section 7, "Other Commands," for details.
- See also "Automatic-Duplicate Modifier [D]" and "Generate Modifier [G]" in this section.
- See also Section 5, "Validation and Generation Descriptors."

Examples

The A modifier in each of these examples is underlined. In all cases, the indicated value is generated into the field if the operator presses AUX DUP at the beginning of the field.

G5A#RECNO

Generates a five-character general field (G5) from the system keyword #RECNO.

I4AF2

Generates a four-character integer field (I4) from the contents of field 2 (F2) of this format.

I5AX(145,7,5)

Generates a five-character integer field (I5) from the contents of index set 145 [X(145,7,5)] starting in column 7 for five spaces.

I8AF1\*38

Generates an eight-character integer field (I8) from the contents of field 1 (F1) multiplied by the constant 38.

G6A"JUL 85"

Generates a six-character general field (G6) from the character string "JUL 85".

BALANCE-TOTAL MODIFIER [B]

Use the B modifier to add the contents of a field to a specified accumulator.

The syntax for the B modifier is:

field <u>B</u> <sub>n</sub>
-----------------------------

where

field is the field specification (type, length, optional field number, optional attribute modifier).

n is the number of the accumulator to be used.

Usage Notes

- If the operator enters a negative value in a B modified left-zero-filled field, the negative value is added to the accumulator (equivalent to subtracting a positive value).
- If the operator moves backward in the batch, or backspaces into a field that has a B modifier, the value of the field is subtracted from the accumulator. If the operator moves forward in the batch, or changes a field that has a B modifier, the value of the field is added back into the accumulator.
- If n is the highest numbered accumulator for the job, it is designated as the batch-balance accumulator for that job. When the contents of the batch-balance accumulator are zero, the batch is in balance; when the accumulator contains any value other than zero, the batch is out of balance.
- Any change made in the balance-total fields automatically changes the batch-balance total. The final balance totals are carried into verify mode with the batch.
- You can use the B modifier with all field types except w and with all other field modifiers.
- The B modifier must follow all other modifiers except X and Z. The B modifier can either precede or follow Z. B can only precede the X modifier. B can precede or follow field validation.



Examples

N6B2

A six-character numeric field (N6) whose contents are to be added to accumulator 2 (B2).

N4RB8

A four-character numeric field (N4) that must be released (R) and whose contents are to be added to accumulator 8 (B8). Note that the B modifier is placed after the R modifier.

#### CHECK-DIGIT MODIFIER [C]

Use the C modifier to verify a check digit entered in the rightmost column of a field.

The syntax for the C modifier is:

fieldC<sub>w,m</sub>

where

field is the field specification (type, length, optional field number, optional attribute modifier).

w is the length of the base number plus the check digit.

m is the modulus number (either 7, 10, or 11).

#### Usage Notes

- The value of w is usually the same as the field length unless the base number is spread over several fields, as in a social security number.
- The base number can span several fields, including automatic-duplicate fields, but should not include the P, B, or @ commands. If you are using attribute spaces and are spanning adjacent fields, no attribute spaces can separate these fields.
- The base number can include the letters A through Z, but only the digit part of the letter when represented as a Hollerith card code is used to compute the check digit. See Appendix B, "Check Digit Calculation," for more information.
- You can use the C modifier with all field types except w and with all other field modifiers.
- To generate a check digit, use the C command. The C command is described in Section 7, "Other Commands."

#### Examples

I10C10,11

Checks the tenth digit (C10) of a ten-character integer field (I10), using modulus 11.

A2 I(-)6C8,7

Spans two fields (a two-character alphabetic field plus a six-character integer field) and checks the eighth digit, using modulus 7.

This example assumes a system that uses attribute spaces. Notice that the attribute space between the two fields has been inhibited (-) to make the fields contiguous.

### AUTOMATIC-DUPLICATE MODIFIER [D]

Use the D modifier to generate data into a field by duplicating the value entered into the same field in the preceding record.

The syntax for the D modifier is:

fieldD

where field is the field specification (type, length, optional field number, optional attribute modifier).

### Usage Notes

- ASD (auto skip duplicate) must be turned on for the D modifier to work.
- To duplicate a value, you must not erase the value from the screen between records.

The D modifier converts all duplicated characters to data, regardless of their original attribute. Use care in erasing the previous screen and in deciding what fields are to be your sources for duplication; otherwise, the wrong data can be written to the disk.

- If necessary, the data entry operator can backspace into the field, one character at a time, to make corrections or change the duplicated data.

When ASD is off, the data entry operator can either enter data into the field or duplicate the field manually by pressing DUP.

- In a new record, if the data entry operator presses ACCEPT before reaching a D-modified field, the D-modified field is blank-filled unless it is combined with a must-enter (E) modifier.
- In key verify mode, the D modifier causes the keyed entry to be compared column by column with the previous record (assuming ASD is on). If you use D on a field to be key-verified, the field must always be in the same position on the screen and in each output record.
- The D modifier does not function in verify reconstruct mode.
- You can use the D modifier with all field types and with all other field modifiers except G and S.
- When you write a format for interactive communications with a host computer, you must use the D modifier on each field to be filled in by the host. Format programming for interactive communications is described in Section 12.



- See also "Auxiliary-Duplicate Modifier [A]" and "Generate Modifier [G]" in this section.

Examples

I5D

A five-character integer field (I5) to be automatically duplicated (D) from the same field position on the previous screen (assuming that ASD is on).

### MUST-ENTER MODIFIER [E]

Use the E modifier to indicate that the data entry operator must enter at least one character in a field.

The syntax for the E modifier is:

fieldE

where field is the field specification (type, length, optional field number, optional attribute modifier).

### Usage Notes

- The data entry operator cannot use the RETURN, ACCEPT or TAB keys in the first column of a must-enter field.
- The data entry operator can use the DUP key anywhere in a must-enter field.
- If the data entry operator presses the ACCEPT key before reaching a must-enter field, the release of the record is stopped at the must-enter field and the message "MUST ENTER FIELD" flashes on the screen.
- You can use the E modifier with all field types except W and with all other field modifiers.

### Examples

I5E

A five-character integer field (I5) in which at least one character must be entered (E) by the operator.

### MUST-FILL MODIFIER [F]

Use the F modifier to indicate that if any data is entered in a field, the entire field must be filled.

The syntax for the F modifier is:

fieldF

where field is the field specification (type, length, optional field number, optional attribute modifier).

### Usage Notes

- Unless the cursor is located on the first column of the field, the data entry operator cannot use the keys RETURN, ACCEPT, or TAB until the field is filled.
- The data entry operator can use the DUP key anywhere in a must-fill field.
- If the F modifier is used in a field with the must-release modifier (R), the data entry operator must fill the field and then release it by pressing the RETURN key.
- You can use the F modifier with all field types except W and with all other field modifiers except J. Left-zero-fill fields are automatically filled; F is legal to use with these fields, but it is ignored.

### Examples

G8F

An 8-character general field (G8) that must be filled (F).

G8FR

An 8-character general field (G8) that must be filled (F) and that must be released (R).

Section 3  
Field Modifiers

GENERATE MODIFIER [G]

Use the G modifier to generate data into a field automatically, with no operator intervention.

The syntax for the G modifier is:

fieldGvalue

where

field is the field specification (type, length, optional field number, optional attribute modifier).

value is either the value to be generated or a source from which the value is to be generated. value can be a numeric constant, an arithmetic expression, a character string, a system keyword, or one of the following validation/generation descriptors:

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c for a length of l characters.

Ff refers to field f of the current format.

LFf refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c for a length of l characters.

RS(n,l) refers to a relative screen position, beginning n columns from the current position for a length of l characters.

S(r,c,l) refers to the screen area beginning in row r, column c, for a length of l characters.

X(id,c,l) refers to index set id, beginning in column c for a length of l characters.

.symbol. refers to a name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

Usage Notes

- ASD must be on for the G modifier to work.



- If necessary, the data entry operator can turn off ASD (by pressing the ASD key) and backspace into the field to make corrections or change the generated data.
- LIfE-Works uses accumulator 0 as a temporary storage area for numeric constants or arithmetic expressions. Therefore, if you are generating from a numeric constant or arithmetic expression, the length of the field cannot exceed the accumulator length (12 columns for single-precision accumulators, 24 columns for double-precision accumulators).
- If the generated value is a character string:
  - The entire string must be enclosed in quotation marks.
  - The string cannot exceed 180 characters.
  - The number of characters in the string must be less than or equal to the number of characters specified in the field type. The format program fills any unused spaces with blanks to the right of the last character.
- A generated field overrides the specified field type. For example, if you specify that a string of digits is to be generated into an alphabetic field, the format program will do so, without producing any error message.
- In a new record, if the data entry operator presses the ACCEPT key before reaching a G modified field, the G modified field is blanked unless it also has an E modifier.
- You can use the G modifier with all field types except W and with all other field modifiers except A, S, and D.
- The G modifier does not function in verify reconstruct mode.
- See also "Auxiliary-Duplicate Modifier [A]" and "Automatic-Duplicate Modifier [D]" in this section.
- See also Section 5, "Validation and Generation Descriptors."
- If the length of the generated value does not match the length of the field, LIfE-Works truncates or pads the data. See the discussion of nonmatching lengths in Section 7, "Other Commands," for details.

Section 3  
Field Modifiers

Examples

The G modifier in each of these examples is underlined for clarity. In each case, the indicated value is generated when the operator moves the cursor into the field.

I12GA3

Generates a 12-character integer field (I12) from accumulator 3 (A3).

G5GS(10,1,5)

Generates a five-character general field (G5) from the contents of the screen (S) starting in row 10, column 1, for five spaces.

G5G10000

Generates a five-character general field (G5) from the constant 10000.

I8GF1\*38

Generates an eight-character integer field (I8) from the contents of field 1 (F1) multiplied by the constant 38.

G6G"JUL 85"

Generates a six-character general field (G6) from the character string "JUL 85".

G15G.NAME.

Generates a 15-character general field (G15) from the location represented by the symbol NAME.

### INDEX-GENERATE MODIFIER [I]

Use the I modifier to generate data into a field from a value set.

The syntax for the I modifier is:

fieldIx

where

field is the field specification (type, length, optional field number, optional attribute modifier).

x indicates which value set is to be used (1-15).

### Usage Notes

- To generate data from a value set, the data entry operator moves the cursor to the first column of the field, presses INDEX, then presses a number or letter key. The keys 1 through 9 correspond to the first 9 items of the value set; the keys A through Z (upper- or lowercase) correspond to the first 26 items of the value set. (Note that only the first 26 items of a value set can be generated.)

If the operator types a data character in the first column of the field instead of pressing INDEX, the field accepts normal data entry. If the operator presses INDEX in any column except the first, LIFE-Works displays an error message.

- You can use the I modifier with all field types except W and with all other field modifiers except A.
- See also Section 9, "Value Sets."

### Examples

G10I2

Generates a ten-character general field (G10) from value set 2 (I2) when the operator presses INDEX and a selection key.



### JUSTIFY MODIFIER [J]

Use the J modifier to right-justify the data in a field.

The syntax for the J modifier is:

fieldJ

where field is the field specification (type, length, optional field number, optional attribute modifier).

### Usage Notes

- If the data entry operator presses RETURN in a partially filled J modified field, the contents of the field are moved to the right until the rightmost column is filled, then the leftmost columns are filled with blanks.
- You can use the J modifier with all field types except W and with all other field modifiers except F.

### Examples

N9J

A nine-character numeric field (N9) that is right-justified (J).

N9JE

A nine-character numeric field (N9) that is right-justified (J) and must-enter (E).



### DISPLAY-ONLY MODIFIER [Q]

Use the Q modifier to indicate that the contents of a field are for display only--that is, the data entered or generated is not to be transferred to the disk.

The syntax for the Q modifier is:

fieldQ

where field is the field specification (type, length, optional field number, optional attribute modifier).

### Usage Notes

- The Q modifier is often used with a generated field for the operator's information or for a working balance.
- Do not place Q modified fields inside conditional logic statements (IF...THEN statements). When existing records are displayed, the screen positions for display-only fields contain blanks; therefore, the results are not predictable.
- You can use the Q modifier with all field types except W and with all other field modifiers.

### Examples

G10Q

A ten-character general field (G10) that displays on the screen but is not stored on disk (Q).

N12GA1Q

A 12-character numeric field (N12) generated (G) from accumulator 1 (A1). The total displays on the screen but is not stored on disk (Q).

Section 3  
Field Modifiers

MUST-RELEASE MODIFIER [R]

Use the R modifier to indicate a must-release field--that is, a field that requires the operator to press RETURN or ACCEPT before continuing to the next field or record.

The syntax for the R modifier is:

fieldR

where field is the field specification (type, length, optional field number, optional attribute modifier).

Usage Notes

- The data entry operator must release a must-release field even if the field is completely filled with data. Use the R modifier to provide positive separation between adjoining fields or to require a positive release at the end of a record.
- The data entry operator can use the DUP key to perform normal duplicate functions in a must-release field.
- You can use the R modifier with all field types except L and W, and with all other field modifiers.

Examples

A10R

A ten-character alphabetic field (A10) that must be released (R).

A10ER

A ten-character alphabetic field (A10) that must be entered (E) and must be released (R).

### SKIP MODIFIER [S]

Use the S modifier to fill a field with blanks when ASD (auto skip duplicate) is on.

The syntax for the S modifier is:

```
fieldS
```

where field is the field specification (type, length, optional field number, optional attribute modifier).

### Usage Notes

- The S modifier is most often used to "pad" a field in an IF...THEN statement to make the screen and record match each other in length. It is typically used with a G field type.
- If ASD is off, the data entry operator can enter data in the normal manner.
- You can use the S modifier with all field types and with all other field modifiers except D and G.

### Examples

```
G10S
```

A ten-character general field (G10) that is filled with blanks (S) if ASD is on.

```
P"Do you like mustard?" 1G1=SA"yn"  
IF F1="y"  
  THEN (P"Enter favorite brand" G15)  
  ELSE (P"          "  
        ASD G15S ASD#)
```

A conditional statement with the ELSE branch padded so it matches the THEN branch. If the respondent doesn't like mustard, there is no reason to answer the question about a favorite brand. The ELSE path provides a 15-character general field to match the THEN branch, and the S modifier fills it with blanks. The ASD command turns ASD on, so the S modifier will work; the ASD# command turns ASD off.



### EXECUTE C FUNCTION MODIFIER [X]

Use the X field modifier to execute a function that has been coded in the C programming language and to pass the value of the modified field to the function as an argument.

The syntax for the X modifier is:

`fieldX(i[,argument,argument...])`

where

field is the field specification (type, length, optional field number, optional attribute modifier).

i is the function index (1-999999). i can be either a numeric constant or one of the following validation/generation descriptors:

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c and extending for l columns.

Ff refers to field f of the current format.

Lff refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c and extending for l columns.

RS(n,l) refers to a relative screen area within a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to the screen area beginning in row r, column c, and extending for l columns.

X(id,c,l) refers to index set id, beginning in column c and extending for l columns.

.symbol. refers to a symbolic name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

argument is an optional argument that is passed to the function as a character string. You can send up to 48 arguments. argument can be any of the following validation/generation descriptors:

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c and extending for l columns.

Ff refers to field f of the current format.

LFf refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c and extending for l columns.

RS(n,l) refers to a relative screen area within a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to a screen area beginning in row r, column c, and extending for l columns.

X(id,c,l) refers to index set id, beginning in column c and extending for l columns.

.symbol. refers to a symbolic name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

#### Usage Notes

- The specified C function must exist when the format is run.
- The value of the field associated with the X modifier is passed as the first argument to the C function. The field value is followed by any specified optional arguments.
- The specified C function can change the value of the field associated with the X modifier. When the function completes execution, the X-modified field is updated and redisplayed on the LiFE-Works screen.

Be aware that the X-modified field is the only field that is automatically redisplayed. If the C function changes the value of any other screen area, the change is not displayed automatically. It is strongly recommended that you make all changes in an offscreen area, such as an accumulator or buffer.

- You can use the X modifier with all field types except w and with all other field modifiers. X must follow all other field modifiers and any field validation.
- The X modifier is ignored in pass 1 and when validation is off. (See Appendix A, "Record Processing," for information on pass 1.)
- To use the X modifier, the \$W/cd directory must be installed on your system. See the LiFE-Works System Release Guide for additional information.
- For a detailed example, see the source module cd.exec.c in the \$W/cd directory.
- See also "Execute C Function Command [EXEC]" in Section 7, "Other Commands."

Section 3  
Field Modifiers

ZERO-BALANCE MODIFIER [Z]

Use the Z modifier to subtract the contents of a field from a specified accumulator.

The syntax for the Z modifier is:

field <u>Z</u> n
------------------

where

field is the field specification (type, length, optional field number, optional attribute modifier).

n is the number of the accumulator to be used.

Usage Notes

- Z is most commonly used to maintain a batch balance total. Always use the highest numbered accumulator for batch balancing.  
  
Load the batch balance into the zero-balancing accumulator at the start of each record or batch. Any changes to a field in entry mode automatically change the zero-balance total accordingly. The final balance should be zero.
- You can use the Z modifier with all field types except w and with all other field modifiers.
- Place the Z modifier after all other modifiers except B and X. The Z modifier can either precede or follow B. Z can only precede the X modifier. Z can precede or follow field validation.
- Do not confuse the Z modifier with the Z command. The Z command is described in Section 7, "Other Commands."









## Section 4 Attributes

### INTRODUCTION

An attribute is an internal code associated with each data entry field or prompt. LIFE-Works uses attributes to distinguish between data, which is stored on disk when a record is released, and prompts, which are not stored with the record.

The format language elements that generate attributes are:

- All types of entry fields (A, a, G, g, H, I, L, Lw.d, and W). All characters in an entry field are treated as data unless the format specifies otherwise.
- T (text) commands. Characters placed on the screen by a T command are treated as data.
- P (prompt) commands. Characters placed on the screen by a P command are treated as prompts.
- B (blank) commands. Blanks placed on the screen by a B command are treated as prompts.
- @ (tab) commands. Characters in the path of an @ command are converted to prompts, regardless of their original attribute.

You can do certain attribute-related operations through format code. These are:

- Alter the appearance of the screen display by using attribute modifiers. The attribute modifiers are:

<u>Modifier</u>	<u>Meaning</u>
H	high intensity (bright)
L	low intensity (dim)
B	blank intensity (invisible)
U	underscored
F	flashing
R	reverse video
Bn	color of background
Cn	color of characters

You can use these attribute modifiers to define the appearance of an individual entry field or prompt, or to change the default display characteristic for an entire format.

- Place an attribute at a specified position on the screen. If your system uses attribute spaces, this feature lets you move the invisible attribute character from row 1, column 1 of the screen, thus freeing that position for other use.

## Section 4 Attributes

- Prevent LiFE-Works from generating an attribute for a specified prompt or data field. If your system uses attribute spaces, this feature lets you eliminate unwanted spaces between fields.

This section explains how to use attribute modifiers and describes the commands listed in Table 4-1.

Table 4-1. Commands Related to Attributes

Command	Description
INTEN	Changes the default display characteristic for an entire format.
ATR	Places an attribute at a specified position on the screen.
- (hyphen)	Inhibits the attribute that would normally be generated for a prompt or entry field.

### General Usage Notes

- There are two styles of LiFE-Works attributes. One style places invisible characters on the screen; each character occupies one column of screen space. The other style stores attribute information offscreen and does not occupy any screen space. You should know which attribute style your system is configured for when you start coding. If necessary, check the LWconfig system options or ask your system administrator.

If your system is configured to use attribute spaces, each attribute occupies one column position on the screen immediately before the prompt or field it modifies. You must allow for the invisible attributes when you plan your screen layout. For example, you must allow six column positions on the screen for a five-character data field.

If your system is not configured to use attribute spaces, you don't have to make any allowances for attributes in your code. For example, you allow five column positions on the screen for a five-character data field.

You can override your system's configured attribute style for a specific format. To do this, include the \$ATTRSPACE subcommand in the \$FORMAT command. See Section 1, "Introduction," for detailed information on \$ATTRSPACE.

- To prevent data from being written to disk, use the Q field modifier (for an individual field) or the NDR command (for an entire record). See Section 3, "Field Modifiers," for information on the Q modifier; see Section 7, "Other Commands," for information on the NDR command.



- You can use multiple attribute modifiers for the same element. For example, you can display a prompt that is both reverse video (R) and underlined (U).

Multiple attribute modifiers are legal anywhere a single modifier is legal. You can use any combination of modifiers except for the B, H, L, or R modifiers, which cannot be used together. For clarity, you can use an optional comma between modifiers.

- The color assignments for the color attribute modifiers (Bn and Cn) are in the termcap file. n is a number from 0 to 8. 0 is the default color as specified in termcap. The legend for the other numbers is in the configuration information display (mode S-C).

The color attribute modifiers have no effect on a monochrome display screen.

- On a system that uses attribute spaces, you can display the attribute characters to see which bits are set in the attribute byte. To display the attribute character, press HELP while the terminal is online with the mainframe. See Section 12 for a table that lists the attribute characters and the corresponding bit settings.



#### MODIFYING THE APPEARANCE OF AN INDIVIDUAL ELEMENT

You can use an attribute modifier to alter the appearance of any element that generates an attribute—that is, anything that the system identifies as either a prompt or data.

The syntax for using an attribute modifier with an individual element varies somewhat, depending on the element. The general syntax is:

element(i)

where

element is any of the following:

- An A, a, G, g, I, L, N, H, or W entry field.
- A text command (T).
- A prompt command (P).
- A blank command (B).
- A tab command (@).

i is an attribute modifier, as follows:

H to display the element at high intensity.

L to display the element at low intensity.

B to make the element invisible (blank).

U to underscore the element.

F to make the element flash.

R to display the element in reverse video.

Bn to specify a background color for the element, where n is a number from 0 to 8.

Cn to specify a character color for the element, where n is a number from 0 to 8.

### Usage Notes

- For specific syntax details, see the following sections:
  - Section 2, "Field Types," for entry fields.
  - Section 6, "Screen Formatting Commands," for P, B, I, and @ commands.
- Do not confuse attribute modifiers with field modifiers. Attribute modifiers can be used with entry fields and with other format code elements. Field modifiers can be used only with entry fields. See Section 3 for information on field modifiers.

### Examples

P(H)"NAME"

Displays the prompt NAME at high intensity (H).

P(H,F)"OUT OF RANGE"

Displays the prompt OUT OF RANGE at high-intensity (H) and makes it flash (F).

G(C7,B5)20

Displays a 20-character general field with characters in color 7 and the background in color 5.

### MODIFYING THE DEFAULT DISPLAY CHARACTERISTIC [INTEN]

Use the INTEN compiler directive to change the default display characteristic of an entire format.

The syntax for the INTEN directive is:

INTEN i

where i is an attribute modifier, as follows:

H for high intensity.

L for low intensity.

B for invisible (blank).

U for underscored.

F for flashing.

R for reverse video.

Bn for a background color, where n is a number from 0 to 8.

Cn for a character color, where n is a number from 0 to 8.

#### Usage Notes

- The default display characteristic at the beginning of any format is high (bright). The default characteristic applies to any prompt or data for which no attribute modifier is specified.
- The INTEN directive changes the default display characteristic for an entire format until another INTEN directive is encountered or until the end of the format is reached. You can use the INTEN directive as many times as you like in a format.
- Like other compiler directives, INTEN takes effect during format compilation rather than during execution. Don't use an INTEN directive in an IF or SPLIT statement; you may not get the intended result.
- For information on changing the appearance of an individual entry field or prompt, see "Modifying the Appearance of an Individual Element" in this section.



Examples

INTEN R

Changes the default display characteristic to reverse video.

INTEN L,B3,C5

Changes the default display characteristic to low intensity, with background color 3 and character color 5.

#### PLACING AN ATTRIBUTE IN A SPECIFIED LOCATION [ATR]

Use the ATR command to place an attribute in a specified row and column. The ATR command is designed primarily for systems that use attribute spaces. It is most often used to move an attribute space out of row 1, column 1 so another character can be placed in that position.

The syntax for the ATR command is:

`ATR(element,i)r,c`

where

element is one of the following:

- An A, a, G, g, I, L, N, H, or W entry field.
- A text command (T).
- A prompt command (P).
- A blank command (B).
- A tab command (@).

i is an attribute modifier, as follows:

H to display the element at high intensity.

L to display the element at low intensity.

B to make the element invisible (blank).

U to underscore the element.

F to make the element flash.

R to display the element in reverse video.

Bn to specify a background color for the element, where n is a number from 0 to 8.

Cn to specify a character color for the element, where n is a number from 0 to 8.

r,c is the row and column in which you want to place the attribute space.

Usage Notes

- If your system uses attribute spaces, LiFE-Works places an invisible attribute in row 1, column 1 of the screen. This attribute, called the field 1 attribute, governs the first element on the screen. If you wish to place a character in row 1, column 1, use the ATR command to move the field one attribute somewhere else on the screen. (It is best to move the attribute to the lower right corner of the screen, where it won't interfere with the rest of the format.)

You can use the ATR command as often as you like in a format program. The last ATR command encountered by the format is the one that governs the field one attribute.

- Be careful when using the ATR command. You can cause data to be lost or changed by mixing up prompt attributes and entry field attributes.

Examples

```
ATR(P,H)22,80
```

Places the field one attribute for a high intensity prompt (P,H) in row 22, column 80.

```
ATR(T,B2,C7)15,1
```

Places the field one attribute for a text string (T) with background color 2 and character color 7 (B2,C7) in row 15, column 1.



#### INHIBITING ATTRIBUTE GENERATION [-]

If your system is configured for attribute spaces, you may want to inhibit attributes for certain elements. For example, lengthy account numbers and dates are often coded as multiple contiguous fields. You can eliminate unwanted spaces between the fields by inhibiting the attributes.

To inhibit an attribute for any given element, use a hyphen where you would normally use an attribute modifier. The general syntax is:

element(-)

where

element is the element whose attribute is to be inhibited. element can be any one of the following:

An A, a, G, g, I, L, N, or H entry field.

A text command (T).

A prompt command (P).

A blank command (B).

A tab command (@).

- is the minus sign (hyphen). The minus sign must be enclosed in parentheses.

#### Usage Notes

- For specific syntax details, see the following sections:
  - Section 2, "Field Types," for entry fields.
  - Section 6, "Screen Formatting Commands," for P, B, T, and @ commands.
- You can inhibit attributes only between fields of the same type; that is, between data fields or between prompt fields.

When you inhibit an attribute for a field, LiFE-works applies the attribute from the previous field to the current field. If one field is a prompt field and the other is a data field, LiFE-Works can't tell whether you want the current field to be part of the record or not. If you try to inhibit attributes between an illegal combination of field types, an error message appears when you compile the program.

- You cannot inhibit the attribute for a W field.

Examples

A4 G(-)28

A four-character alphabetic field with the default attribute followed by a 28-character general field with the attribute inhibited.

I2 T(-)"/" I(-)2 T(-)"/" I(-)2

A date field in which the attribute spaces have been inhibited between the parts of the date. An entry in this field would display as 06/25/85 rather than 06 / 25 / 85.









Section 5  
Validation and Generation Descriptors

INTRODUCTION

Validation is a format programming technique by which data can be checked for validity as it is entered by the operator. For example, newly entered data can be checked against a list of valid items in a value set.

Generation is a format programming technique by which data can be copied, or generated, from another source. For example, the contents of a field can be generated from an accumulator.

Validation and generation reduce data entry errors. Generation also reduces the keystrokes required to enter repetitive data.

Validation/generation descriptors supply the values to be validated against or generated from. Validation/generation descriptors fall into three general categories: location descriptors, system keywords, and constant descriptors.

- A location descriptor lets you validate against or generate from a value in a specified location. For example, the A validation/generation descriptor lets you validate against the contents of an accumulator or generate data from the contents of an accumulator.
- A system keyword lets you validate against or generate from an internal system value. For example, the #JOB system keyword lets you validate entered data against the current job name or generate the current job name into a data field.
- A constant descriptor lets you validate against or generate from a constant value. For example, the text descriptor lets you validate against or generate from a specified character string, and the SA descriptor lets you validate against a specified set of valid characters.

Table 5-1 describes the validation/generation descriptors.

Table 5-1. Validation/Generation Descriptors

Descriptor	Description
A	Validate against or generate from a specified accumulator.
B	Validate against or generate from a specified portion of the the work buffer.
#BATCH	Validate against or generate from the current batch identifier.
#CONST	Validate against or generate from the current system constant.
#DATE	Validate against or generate from the current date, in the nine-character format MM/DD/YY.
#DATEL	Validate against or generate from the current date, in the 12-character format MMM DD, YEAR.
#DAY	Validate against or generate from the Julian date (three numeric digits representing the day of the year).
F	Validate against or generate from a specified field in the current record.
#JOB	Validate against or generate from the current job name.
LF	Validate against or generate from a specified local field (a field within the current subroutine).
?LJNIB	Validate by checking for left-justification with no imbedded blanks.
?NEG	Validate by checking for negative-overpunch.
#OPER	Validate against or generate from the current operator's LiFE-Works operator ID.
Rfn	Validate against or generate from a specified portion of the record buffer for MSFP logical file <u>fn</u> . The Rfn descriptor is described in the <u>LiFE-Works Format Language Manual, Volume 2</u> .
#RECNO	Validate against or generate from the current record number.



Table 5-1. Validation/Generation Descriptors (Continued)

Descriptor	Description
RS	Validate against or generate from the contents of a specified relative screen area (a screen area relative to the cursor position when the current subroutine was called).
S	Validate against or generate from the contents of a specified screen area.
SA	Validate by checking each character in the data against a specified set of valid characters (scan-for-any).
SN	Validate by checking each character in the data against a specified set of invalid characters (scan-for-none).
#TERMNO	Validate against or generate from the current terminal number.
#TIMES	Validate against or generate from the system time of day.
#USERID	Validate against or generate from the operator's operating-system user ID.
V	Validate by checking against the items in a specified value set.
X	Validate against or generate from a specified portion of an index set. The X descriptor is described in Section 10, "Index Sets."
.symbol.	Validate against or generate from a name (symbol) that has been assigned to a screen area or a field in an index set record, work buffer, or MSFP record buffer.
"text"	Validate against or generate from a specified character string (string constant).
\	Validate against or generate from a specified hexadecimal constant.

NOTE

See Section 10, "Index Sets," for validation/generation descriptors used exclusively with index sets.

Section 5  
Validation and Generation Descriptors

General Usage Notes

- You can use location descriptors with the SET command to copy values from one location to another. For example, the statement

SET S(2,1,10)=B(1,10)

copies data from the work buffer (B) to the screen (S). You can also use location descriptors to specify a job and batch name for a BRANCH, SUPERS, or SUPERM format command.

- You can shorten a system keyword value as follows:
  - To eliminate characters from the right side of a keyword value, divide the keyword by a power of 10. Each zero in the divisor eliminates one character from the value. For example, if the value of #CONST is CALIFORNIA, the value of #CONST/100 is CALIFORNIA.
  - To eliminate characters from the left side of a keyword value, generate the value into a field that is shorter than the value. For example, if the value of #DATEL is JAN 01, 1987, generating a four-column field from #DATEL (G4G#DATEL) produces the value 1987.

You cannot use the SET command to set the value of any system keyword discussed in this section.

Notes on validation:

- To validate a field against a value, use the following syntax:

$$\text{field} \left\{ \begin{array}{c} = \\ ! \\ < \\ > \end{array} \right\} \text{value}$$

where

field is one of the following:

- a field specification (type, length, optional field number, and optional field and attribute modifiers). Example: 1G25
- a reference to a previously defined field. Example: F2

value is one of the following:

- a validation/generation descriptor. Example: X(111,5,20)

- an arithmetic expression using any logical combination of validation/generation descriptors and numeric constants. Example: A1+10.

For example, the statement

I6=F3+F4

validates a 6-character integer field (I6) against the sum of field 3 (F3) and field 4 (F4).

- LfE-Works uses the standard ASCII character set collating sequence for "less than" or "greater than" comparisons (< or >). For example, all uppercase letters are greater than lowercase letters, and all alphabetic characters are greater than all numbers.
- Use the logical symbols & and | to indicate AND and OR between validation/generation descriptors. For example, the statement

I3=V1|=V2

validates a three-digit integer field against the items in either value set 1 or value set 2. On the other hand, the statement

I3!V1&!V2

indicates that the integer field is valid if it does not contain any item in either value set.

The symbol & is less commonly used because AND is implied between expressions that don't have any other symbols. For example, the above statement could also be written as

I3!V1!V2

The | symbol has a different function when it appears inside a quoted string constant; in this case, it indicates column positions that are not to be validated. See "String Constant Validation/Generation Descriptor [text]" in this section for details.

- Validation does not occur under the following conditions:
  - When the illegible-field key is used in a record.
  - When validation override is in effect (after the VALID key is pressed).
  - When RETURN or ACCEPT is pressed in the first position of a field (except when that field is a must-enter field).
  - During key verify (unless one or more characters in the field have been changed).



Section 5  
Validation and Generation Descriptors

- See Section 8, "Arithmetic and Logic," for information on arithmetic expressions using validation/generation descriptors.

Notes on generation:

- To generate a value into a field, use a statement of the form:

$$\text{field} \left\{ \begin{matrix} A \\ G \end{matrix} \right\} \text{value}$$

where

field is a field specification (type, length, optional field number, and optional field and attribute modifiers). Example: 3A10

A is the auxiliary-duplicate field modifier (see Section 3).

G is the generate field modifier (see Section 3).

value is one of the following:

- a validation/generation descriptor.
- an arithmetic expression using any logical combination of validation/generation descriptors and numeric constants. Example: A1+10.

For example, the statement

I6GF4+1

adds the constant 1 to the value of field 4 (F4+1) and generates (G) the resulting value into a six-digit integer field (I6).

- See Section 8, "Arithmetic and Logic," for information on arithmetic expressions using validation/generation descriptors.



### ACCUMULATOR VALIDATION/GENERATION DESCRIPTOR [A]

Use the A descriptor to validate against or generate from a specified accumulator.

The syntax for the A descriptor is:

An

where n is the number of the accumulator to be used.

#### Usage Notes

- The value of n can be 0 through 23, but cannot exceed the number of accumulators specified by \$ACCUMS in the \$DEFJOB statement for the current job.
- Use consecutive accumulators within a job. For example, if you have defined three accumulators, use accumulators 1, 2 and 3.
- Accumulator 0 is generally reserved for system use (for generation or validation of fields) except in certain situations. See Section 8, "Arithmetic and Logic," for details.
- See also Section 8 for details on using accumulators in arithmetic expressions and logic statements.

#### Examples

I8=A3+A4

Validates an 8-character integer field (I8) against the value produced by adding the contents of accumulator 3 (A3) to the contents of accumulator 4 (A4).

I6GA3

Generates (G) a 6-character integer field (I6) from the contents of accumulator 3 (A3).

WORK BUFFER VALIDATION/GENERATION DESCRIPTOR [B]

Use the B descriptor to validate against or generate from the current terminal's work buffer.

The syntax for the B descriptor is:

B(c,l)

where

c specifies the beginning column in the work buffer. Any entry between 1 and 2000 is valid.

l specifies the length of the field in the work buffer. Any entry between 1 and 2000 is valid.

Usage Notes

- Use the GETBUF format command to allocate a work buffer, and use the RELBUF command to release a work buffer. GETBUF and RELBUF are described in Section 7, "Other Commands."

Examples

G18=B(36,18)

Validates an 18-character general field (G18) against the contents of the work buffer (B) starting in column 36 for a length of 18 characters.

A20GB(23,20)

Generates a 20-character alphabetic field (A20) from the contents of the work buffer (B) starting in column 23 for a length of 20 characters.

BATCH IDENTIFIER SYSTEM KEYWORD [#BATCH]

Use the #BATCH system keyword to validate against or generate from the current batch name.

The syntax for the #BATCH keyword is:

#BATCH

Usage Notes

- The current batch name is displayed on the status line while the format is running.
- #BATCH is six characters long and is left-justified with trailing blanks.

Examples

G6=#BATCH

Validates a 6-character general field (G6) against the current batch name.

G6G#BATCH

Generates (G) a 6-character general field (G6) from the current batch name.

G3G#BATCH/1000

Truncates three characters from the right side of #BATCH (#BATCH/1000) and generates (G) the remaining characters into a three-character field (G3). For example, if the current batch name is FRIDAY, this statement generates FRI into the field.



Section 5  
Validation and Generation Descriptors

SYSTEM CONSTANT SYSTEM KEYWORD [#CONST]

Use the #CONST system keyword to validate against or generate from the current system constant.

The syntax for the #CONST keyword is:

#CONST

Usage Notes

- The system constant value is set during configuration or through the \$CONST supervisory command.
- #CONST is a 12-character value and is left-justified with trailing blanks.

Examples

G12=#CONST

Validates a 12-character general field (G12) against the system constant.

G3G#CONST/1000000000

Truncates nine characters from the right side of #CONST (#CONST/1000000000) and generates (G) the remaining characters into a three-character field (G3). For example, if the system constant is NOVEMBER~~0000~~, this statement generates NOV into the field.



NUMERIC DATE SYSTEM KEYWORD [#DATE]

Use the #DATE system keyword to validate against or generate from the current date. If your system configuration uses European dates, the format is DD/MM/YYØ; otherwise, the format is MM/DD/YYØ.

The syntax for the #DATE keyword is:

#DATE

Usage Notes

- The date is maintained by the system clock. It increments daily at midnight.
- #DATE is a nine-character value and is left-justified with a trailing blank.

Examples

I3G#DATE

Generates (G) a 3-character integer field (I3) from the rightmost characters in the current date. For example, if today's date is 11/01/87, this statement generates the characters 87Ø.

N5G#DATE/10000

Truncates four characters from the right side of #DATE (#DATE/10000), then generates (G) the remaining characters into a five-character field (N5). For example, if today's date is 11/01/87, this statement generates 11/01 into the field or 01/11 if your system uses European dates.

Section 5  
Validation and Generation Descriptors

ALPHABETIC DATE SYSTEM KEYWORD [#DATEL]

Use the #DATEL system keyword to validate against or generate from the current date. If your system configuration uses European dates, the format is DD MMM, YYYY; otherwise, the format is MMM DD/YYYY.

The syntax for the #DATEL keyword is:

#DATEL

Usage Notes

- The date is maintained by the system clock. It increments daily at midnight.
- #DATEL is a 12-character value and is left-justified.

Examples

G12=#DATEL

Validates a 12-character general field (G12) against the current date.

G12G#DATEL

Generates (G) a 12-character general field (G12) from the current date.

G4G#DATEL

Generates (G) the rightmost four characters of #DATEL into a four-character general field. For example, if today's date is August 21, 1987, this statement generates 1987 into the field.

G6G#DATEL/1000000

Truncates six characters from the right side of #DATEL (#DATEL/1000000), then generates (G) the remaining characters into a six-character general field (G6). For example, if today's date is October 8, 1987, this statement generates OCT 08 into the field.

Section 5  
Validation and Generation Descriptors

DAY SYSTEM KEYWORD [#DAY]

Use the #DAY system keyword to validate against or generate from the system Julian date (three-digit day of the year).

The syntax for the #DAY keyword is:

#DAY

Usage Notes

- The Julian date is maintained by the system clock. It increments daily at midnight.
- #DAY is a three-digit value and is right-justified with leading zeroes.

Examples

I3E=#DAY

Validates a three-digit, must-enter integer field (I3E) against the current Julian date.

I3G#DAY

Generates (G) a three-digit integer field (I3) from the current Julian date.



FIELD REFERENCE VALIDATION/GENERATION DESCRIPTOR [F]

Use the F descriptor to validate against or generate from a specified field in the current record.

The syntax for the F descriptor is:

Ff
----

where f is the number of the field to be referenced.

Usage Notes

- The field you are referring to must have a field number associated with it. (For example, 1G20 assigns the field number 1 to a 20-character general field.) See Section 2, "Field Types," for more information on field numbers.
- When you set a value into a field by using the SET command, you can specify whether the characters in the field are to be treated as data or as prompts.

Use the D modifier with the F descriptor to ensure that the characters in the field are treated as data. The syntax is:

FDf

where f is the field number.

Use the P modifier with the F descriptor to ensure that the characters in the field are treated as prompts. The format is:

FPf

where f is the field number.

The FDf and FPf forms of the F descriptor are valid only on the left side of a SET statement.

If you use the F descriptor on the left side of a SET statement without specifying D or P, LIfE-Works handles the characters as follows:

- On a system that is not configured for attribute spaces, the characters are treated as data.

Section 5  
Validation and Generation Descriptors

- On a system that is configured for attribute spaces, the attribute that precedes the specified field determines whether the characters become data or prompts.

Examples

I4GF2

Generates a 4-character integer field (I4) from the contents of field 2 (F2).

SET FP2="HOURS"

Generates the string constant "HOURS" into field 2 and ensures that the characters are treated as prompts (FP2).

SET FD4=#DATE

Generates the value of the system keyword #DATE into field 4 and ensures that the characters are treated as data (FD4).

JOB NAME SYSTEM KEYWORD [#JOB]

Use the #JOB system keyword to validate against or generate from the current job name.

The syntax for the #JOB keyword is:

#JOB

Usage Notes

- The current job name appears on the status line while the format is running.
- #JOB is nine characters long and is left-justified with trailing blanks.

Examples

G9=#JOB

Validates a 9-character general field (G9) against the current job name.

G9G#JOB

Generates (G) a 9-character field (G9) from the current job name.

G3G#JOB/1000000

Truncates six characters from the right side of #JOB (#JOB/1000000), then generates (G) the remaining characters into a three-character general field (G3). For example, if the value of #JOB is PAYROLL, this statement generates PAY into the field.



LOCAL FIELD REFERENCE VALIDATION/GENERATION DESCRIPTOR [LF]

Use the LF descriptor to validate against or generate from a specified local field--that is, a field within the current subroutine.

The syntax for the LF descriptor is:

LFf
-----

where f is the number of the local field.

Usage Notes

- The LF descriptor is similar to the F descriptor, but LF is used only to identify fields in subroutines. See "Define Subroutine Command [SUBROUTINE]" in Section 7, "Other Commands," for information on fields in subroutines.
- When you set a value into a local field by using the SET command, you can specify whether the characters in the field are to be treated as data or as prompts.

Use the D modifier with the LF descriptor to ensure that all characters in the specified field are treated as data. The syntax is:

LFDf

where f is the field number.

Use the P modifier with the LF descriptor to ensure that all characters in the specified field are treated as prompts. The format is:

LFPf

where f is the field number.

The LFDf and LFPf forms of the LF descriptor are valid only on the left side of a SET statement.

If you use the LF descriptor on the left side of a SET statement without specifying D or P, LiFE-Works handles the characters as follows:

- On a system that is not configured for attribute spaces, the characters are treated as data.



- On a system that is configured for attribute spaces, the attribute that precedes the specified field determines whether the characters become data or prompts.

Examples

I5GLF3

Generates a five-character integer field (I5) from the contents of local field 3 (LF3).

SET LFP2=B(1,10)

Sets local field 2 (LFP2) equal to the contents of columns 1 through 10 of the work buffer (B(1,10)) and ensures that the characters are treated as prompts.

Section 5  
Validation and Generation Descriptors

LEFT-JUSTIFIED WITH NO IMBEDDED BLANKS VALIDATION DESCRIPTOR [?LJNIB]

Use ?LJNIB to check that entered data is left-justified and does not contain any imbedded blanks.

The syntax for the ?LJNIB descriptor is:

?LJNIB

Usage Notes

- ?LJNIB can be used only for validation and only with the = sign or the ! sign. To pass validation when = is used, the field cannot begin with a blank unless the entire field is blank, and it cannot contain blanks between characters. However, blanks can follow the last character entered.
- Use ?LJNIB with the ! sign in an IF clause to produce custom error messages. (See Section 8, "Arithmetic and Logic," for details.)

Examples

G5=?LJNIB

Validates a five-character general field (G5) by checking that the data is left justified and does not contain embedded blanks. For example:

<u>Pass</u>	<u>Fail</u>
ABXXX	XABXX
XXXXX	AMXXX

If the operator tries to enter a blank as the first character or to type a blank within the data, the standard error message "NOT VALID DATA" appears.

G5 IF!?LJNIB THEN "NO BLANKS ALLOWED IN THIS DATA"

Validates a 5-character general field (G5) by checking that the data is left justified and does not contain imbedded blanks. If the operator tries to enter a blank as the first character or to type a blank within the data, the custom error message "NO BLANKS ALLOWED IN THIS DATA" appears.

NEGATIVE-OVERPUNCH CHARACTER VALIDATION DESCRIPTOR [?NEG]

Use the ?NEG descriptor to validate data by checking for a negative-overpunch character in the rightmost position.

The syntax for the ?NEG keyword is:

?NEG

Usage Notes

- ?NEG can be used only for validation and only with the = sign or the ! sign. To pass validation when = is used, the data must contain a negative overpunch value (J, K, L, M, N, O, P, Q, R, or }) in the rightmost position. To pass validation when ! is used, the data must not contain a negative overpunch value in the rightmost position.
- Use ?NEG with either the = sign or the ! sign to produce custom error messages. (See Section 8, "Arithmetic and Logic," for details.)
- For more information on negative overpunch values, see the descriptions of the left-zero-fill field types (L and Lw.d) in Section 2, "Field Types."

Examples

IF F1=?NEG THEN F3

If field one (F1) has a negative overpunch character in the rightmost position, goes to program level 3 (F3).

IF A1!?NEG THEN "TOTAL IS ZERO OR GREATER"

Displays the custom error message TOTAL IS ZERO OR GREATER if accumulator one (A1) does not have a negative overpunch character in the rightmost position.



OPERATOR ID SYSTEM KEYWORD [#OPER]

Use the #OPER system keyword to validate against or generate from the current operator's LiFE-Works operator ID.

The syntax for the #OPER keyword is:

#OPER

Usage Notes

- The current LiFE-Works operator ID appears on the status line while the format is running.
- #OPER is three alphabetic characters and is right-justified.
- To validate against or generate from the nine-character operating-system logon ID, use the #USERID system keyword. #USERID is described in this section.

Examples

A3E=#OPER

Validates a 3-character must-enter alphabetic field (A3E) against the current LiFE-Works operator ID.

A3G#OPER

Generates (G) a 3-character alphabetic field (A3) from the current LiFE-Works operator ID.



RECORD NUMBER SYSTEM KEYWORD [#RECNO]

Use the #RECNO system keyword to validate against or generate from the current record number.

The syntax for the #RECNO keyword is:

#RECNO

Usage Notes

- The current record number appears on the status line while the format is running.
- #RECNO is six digits long and is right-justified with leading zeros.

Examples

I6=#RECNO

Validates a 6-character integer field (I6) against the current record number.

I6G#RECNO

Generates (G) a 6-character integer field (I6) from the current record number.

RELATIVE SCREEN POSITION VALIDATION/GENERATION DESCRIPTOR [RS]

Use the RS descriptor to validate against or generate from a relative screen area--that is, a screen location that is a specified distance from the position of the cursor when the subroutine is called.

The syntax for the RS descriptor is:

RS(n,l)
---------

where

n is the number of columns from the initial cursor position (the cursor position when the subroutine was called) to the start of the screen area to be referenced.

l is the length of the screen area to be referenced.

Usage Notes

- The RS descriptor is similar to the S descriptor, but RS is used only to identify screen areas within subroutines.
- When you set a value into a relative screen position by using the SET command, you can specify whether the displayed characters are to be treated as data or as prompts.

Use the D modifier with the RS descriptor to ensure that all characters in the specified screen position are treated as data. The syntax is:

RSD(n,l)

where n is the relative column number (counting from the cursor position at the start of the subroutine) and l is the length of the screen area.

Use the P modifier with the RS descriptor to ensure that all characters in the specified screen position are treated as prompts. The syntax is:

RSP(n,l)

where n is the relative column number (counting from the cursor position at the start of the subroutine) and l is the length of the screen area.

- The RSD and RSP forms of the RS descriptor are valid only on the left side of a SET statement.

If you use the RS descriptor on the left side of a SET statement without specifying D or P, LIfE-Works handles the displayed characters as follows:

- On a system that is not configured for attribute spaces, the characters are treated as data.
- On a system that is configured for attribute spaces, the attribute that precedes the specified screen area determines whether the characters become data or prompts.
- For general information on subroutines, see "Define Subroutine Command [SUBROUTINE]" in Section 7, "Other Commands."
- See also "Set Command [SET]" in Section 7, "Other Commands."

#### Examples

```
I6=RS(12,6)
```

Validates a six-character integer field (I6) against the contents of the screen area starting 12 columns from the cursor position when the current subroutine was called, and extending for six columns (RS(12,6)).

```
SET RS(1,8)=F3
```

Sets the screen area beginning at the cursor position when the current subroutine was called, and extending for eight columns (RS(1,8)), equal to the contents of field 3 in the main program.



SCREEN VALIDATION/GENERATION DESCRIPTOR [S]

Use the S descriptor to validate against or generate from the contents of a specified screen area.

The syntax for the S descriptor is:

$S(r,c,l)$

where

r is the row in which the screen area begins.

c is the column in which the screen area begins.

l is the length of the screen area in characters.

Usage Notes

- When you set a value into a screen area by using a SET command, you can specify whether the displayed characters are to be treated as data or prompts.

Use the D modifier with the S descriptor to ensure that all characters in the specified screen area are treated as data. The syntax is:

$SD(r,c,l)$

where r is the beginning row, c is the beginning column, and l is the length of the screen area.

Use the P modifier with the S descriptor to ensure that all characters in the specified screen area are treated as prompts. The syntax is:

$SP(r,c,l)$

where r is the beginning row, c is the beginning column, and l is the length of the screen area.

The SD and SP forms of the S descriptor are valid only on the left side of a SET statement.

If you use the S descriptor on the left side of a SET statement without specifying D or P, LIfE-Works handles the validated/generated characters as follows:



- On a system that is not configured for attribute spaces, the characters are treated as data.
- On a system that is configured for attribute spaces, the attribute that precedes the specified screen area determines whether the characters are treated as data or prompts.
- If your system is configured for attribute spaces, use caution when using the S descriptor with the SET statement—you could destroy attribute spaces in existing format programs.
- See also "Set Command [SET]" in Section 7, "Other Commands."

Examples

N6=S(2,3,6)

Validates a 6-character numeric field (N6) against the contents of the screen starting in row 2, column 3, for a length of six characters [S(2,3,6)].

N6GS(2,3,6)

Generates (G) the value of a six-character numeric field (N6) from the contents of a screen area that starts in row 2, column 3, and is six columns long [S(2,3,6)].

SET SP(2,3,6)=F5

Places the value of field 5 (F5) on the screen, beginning in row 2, column 3, for a length of six columns, and ensures that the characters are treated as prompts [SP(2,3,6)].

SCAN-FOR-ANY-VALIDATION DESCRIPTOR [SA]

Use the SA descriptor to validate entered data by checking each character against a specified set of characters. The data passes validation if each character is one of the specified characters.

The syntax for the SA descriptor is:

SA"xxx..."

where

"xxx..." is the set of characters to be validated against. The entire set must be enclosed in quotation marks.

Usage Notes

- SA can be used only for validation and only with the = sign or ! sign. To pass validation when = is used, each column position of the data must contain one of the specified characters.
- Use SA with the ! sign in an IF clause to produce a custom error message for the operator. (See Section 8, "Arithmetic and Logic," for details.)
- To validate a 1-character field, it is more efficient to use the SA (or SN) descriptor than to use a value set validation or a sequence of OR tests. For example, use

G1=SA"ABC"

instead of

G1="A"|= "B"|= "C"

or instead of

G1=V1

where V1 contains A, B, and C.

Examples

G5=SA"01."

Validates a 5-character general field (G5) by scanning for a zero, one, or decimal point in each position (SA"01."). Data that contains any other characters fails the validation. For example:

<u>Pass</u>	<u>Fail</u>
11100	ABCDE
.....	1.2.1
1111.	00002

G5!SA"01."

Validates a five-character general field (G5) by scanning for a zero, one, or decimal point in each position (SA"01."). Data that contains only these characters fails the validation (!). For example:

<u>Pass</u>	<u>Fail</u>
ABCDE	11100
1.2.1	.....
00002	1111.

IF I6!SA"01" THEN "MUST CONTAIN 0 OR 1"

Validates a six-character integer field (I6) by scanning for a zero or one in each position. If the operator tries to enter any other characters, the custom error message MUST CONTAIN 0 OR 1 appears on the operator's screen.

### SCAN-FOR-NONE VALIDATION DESCRIPTOR [SN]

Use the SN descriptor to validate data by checking each character against a specified set of characters. The data fails validation if it contains any character in the specified set.

The syntax for the SN descriptor is:

SN"xxx..."

where "xxx..." is the set of characters to be validated against. The entire set must be enclosed in quotation marks.

### Usage Notes

- The SN descriptor can be used only for validation and only with the = sign or the ! sign. To pass validation when = is used, the field being checked must not contain any of the specified characters.
- Use SN with the ! sign to create a custom error message for the operator. See Section 7, "Arithmetic and Logic," for details.
- See also "Scan-for-Any Validation Descriptor [SA]" in this section.

### Examples

G5=SN"01"

Validates a 5-character general field (G5) by scanning for a zero or a one in each position (SN"01"). Data containing either character in any position fails the validation. For example:

<u>Pass</u>	<u>Fail</u>
ABCDE	11100
.....	ABCD1



G5!SN"01"

Validates a 5-character general field (G5) by scanning for a zero or a one in each position (SN"01"). Only data that contains either a zero or a one in at least one position passes the validation. For example:

<u>Pass</u>	<u>Fail</u>
ABCD1	ABCDE
11100	.....

IF I6!SN"01" THEN "CANNOT CONTAIN 0 OR 1"

Validates a 6-character integer field (I6) by scanning for either a zero or a one in each character position. If the operator tries to enter data that contains zero or one, the data fails the validation, and the custom error message CANNOT CONTAIN 0 OR 1 appears on the screen.

Section 5  
Validation and Generation Descriptors

TERMINAL NUMBER SYSTEM KEYWORD [#TERMNO]

Use the #TERMNO system keyword to validate against or generate from the current terminal number.

The syntax for the #TERMNO keyword is:

#TERMNO
---------

Usage Notes

- #TERMNO is 3 digits long and is right-justified with leading zeros.

TIME SYSTEM KEYWORD [#TIMES]

Use the #TIMES system keyword to validate against or generate from the current time of day in the form ØHH:MM:SS (hours:minutes:seconds).

The syntax for the #TIMES keyword is:

#TIMES

Usage Notes

- The time is maintained by the system clock.
- #TIMES is nine characters long and is right-justified and zero-filled.
- #TIMES is typically used for generation, but can also be used for validation.

Examples

G9G#TIMES

Generates (G) a nine-character general field (G9) from the current time of day.

G5G#TIMES/1000

Truncates three characters from the right side of #TIMES (#TIMES/1000), then generates the next five characters into a general field. For example, if #TIMES has the value Ø10:34:16, this statement generates the value 10:34. Note that the leading blank is eliminated by generating into a "short" field (five columns rather than six).

Section 5  
Validation and Generation Descriptors

USER ID SYSTEM KEYWORD [#USERID]

Use the #USERID system keyword to validate against or generate from the current operating-system logon ID.

The syntax for the #USERID keyword is:

#USERID

Usage Notes

- #USERID is a nine-character field and is left-justified with trailing blanks.
- Use the #OPER system keyword to validate against or generate from an operator's three-character Life-Works ID. #OPER is described in this section.

Examples

A9=#USERID

Validates a nine-character alphabetic field against the current operator's system logon ID.

A9G#USERID

Generates (G) a nine-character alphabetic field (A9) from the current operator's system logon ID.



### VALUE SET VALIDATION DESCRIPTOR [V]

Use the V descriptor to validate data by checking the data against a list of items in a value set.

The syntax for the V descriptor is:

Vx
----

where x is the number of the value set to be used (1-15).

### Usage Notes

- The V descriptor can be used only for validation.
- The V descriptor must be used with either the equal sign (=) or the not equal sign (!). If you use the equal sign with the V descriptor, the data being checked passes validation if it matches one of the items in the value set. If you use the not equal sign with the V descriptor, the data being checked passes validation if it does not match any of the items in the value set.
- When a matching item is found during validation, the system loads the number of the matching item (not the value) into accumulator 0. Accumulator zero can then be referred to in a SPLIT statement to cause a logical branch. (See Section 9, "Value Sets," for details of using SPLIT statements with value sets.)
- If most of the values within a defined range are valid entries, it is more efficient to create a value set that lists the invalid entries, then validate against this value set by using the not-equal (!) relational operator. For example, assume a two-digit transaction code field in which ten of the possible 100 entries are not valid codes. It is faster to compare an entered value against ten invalid codes than to compare it against 90 valid codes.
- Large value sets are not efficient. If possible, divide a large value set into two or more value sets that can be selected using conditional logic. For example, instead of validating against a value set that contains several hundred items, use a statement like this:

I3<"500"=V1|>"499"=V2

This statement validates a three-digit entry against value set 2 if the entry is less than 500, or against value set 3 if the entry is greater than 499.

Even though this method adds extra logic to the program, it is faster than searching a large value set.

Section 5  
Validation and Generation Descriptors

- When you are validating a field against a value set and one or more string constants, AND is implied; OR is not allowed. For example, the statement

G4=V1!"X!"Y"

specifies that a four-character general field must equal one of the items in value set one AND that the first character of the field cannot be an X AND cannot be a Y. You can include the AND symbol (&) between each string constant, but it is not required.

- In certain situations, other validation methods are more efficient than value set validation.

If you are validating a one-character field, use a field scan (SA or SN) instead of a value set. For example, instead of validating against a value set that contains the items A, B, and C, use the statement:

G1=SA"ABC"

If you are validating against a consecutive series of values, use range checks and string constants instead of a value set. For example, instead of validating against a value set that contains the items 12, 13, 14, and 15, use the statement:

I2=<"16">"11"

- You can avoid using lengthy value set validation if the expected value has a characteristic that can be detected using AND or OR tests. For example, if any even number between 0 and 998 is a valid entry, you can use the following statement for validation:

I3="||2"|="||4"|="||6"|="||8"|="||0"

You could apply a similar test to validate odd numbers or numbers divisible by 5.

- See Section 9, "Value Sets," for information on creating value sets and on using value set validation with branching logic paths.

Examples

I3=V2

Validates a three-character integer field (I3) against each item in value set 2 (V2) of the current job. The data in the field passes validation if it matches any item in the value set.

SYMBOLIC REFERENCE VALIDATION/GENERATION DESCRIPTOR [.symbol.]

Use the symbol descriptor to validate against or generate from a name (symbol) that has been assigned to a location.

The syntax for the symbol descriptor is:

.symbol.

where symbol is the assigned symbol.

Usage Notes

- A symbol can be assigned to any of the following locations:
  - A screen area.
  - A portion of an index set record.
  - A portion of a work buffer.
  - A portion of an MSFP record buffer.

To use symbols, you must first assign the symbols to locations at the beginning of the format. You can then use the assigned symbols as validation/generation descriptors in format code. See Section 11, "Assigning Symbolic References," for detailed information on assigning symbols.

Examples

G10=.NAME.

Validates a ten-character general field (G10) against the location represented by the symbol NAME.

N12G.TOTAL.

Generates (G) a 12-character numeric field (12G) from the location represented by the symbol TOTAL.



STRING CONSTANT VALIDATION/GENERATION DESCRIPTOR ["text"]

Use the text descriptor to validate against or generate from a specified character string.

The syntax for the text descriptor is:

"text"

where "text" is the character string to be validated against or generated from. The string can contain any characters and can be up to 180 characters long. The entire string must be enclosed in quotation marks.

Usage Notes

- When you validate against a string constant, LIfe-Works compares the characters enclosed in quotes against corresponding characters in the field, starting at the left side of the field.
- Use the < and > relationals with a string constant to range-check data. For example, the statement

I2EF<"13">"00"

specifies a two-character integer field (I2) that will accept any integer less than 13 (<"13") but greater than 00 (>"00").

- You can use the | symbol, enclosed in quotes, to specify columns to be ignored during string comparison. The | symbol holds the place of the ignored columns during the comparison. For example, the statement

I4="||||9"

indicates that any four-digit number that ends in 9 is a valid entry for this field.

- If you are validating a numeric field, it is more efficient to validate against a string constant (numbers contained in quotes) than against a numeric constants (numbers written without quotes).

For example, the statement

I3>199

causes the numeric constant 199 to be placed in accumulator 0 (right-justified with leading zeros). The rightmost three digits of the accumulator are then compared to the contents of the field (I3). In this operation, the digits are processed twice, requiring substantial processing time.



On the other hand, if you code the same statement as

```
I3>"199"
```

the accumulators are not used and the string constant is processed only once. In fact, if the test normally passes, only the first column is actually tested. That is, if a valid entry must be greater than 199 (and most entries are expected to fall into this category) any entry whose first digit is greater than 1 will pass on the first column without testing the remaining columns. Therefore, it is possible to code this statement as `I3>"1"`.

String constant tests are left-justified in the field and numeric constant tests are right-justified. If necessary, use leading zeros in string constants to get proper alignment.

- When you generate from a string constant, the string length must be equal to or less than the field length. If less than the length of the field, the character string is left-justified.
- See also "Generate Modifier [G]" in Section 3, "Field Modifiers."

#### Examples

```
G5="ABC"
```

Validates the first three characters of a five-character general field against the string constant ABC; the last two characters in the field are not checked. The field passes validation if the first three characters are ABC.

```
G5!"ABC"
```

Validates the first three characters of a five-character general field against the string constant ABC; the last two characters in the field are not checked. The field passes validation if the first three characters are not ABC.

```
G5>"123"
```

Validates the first three characters of a five-character general field (G5) against the string constant 123; the last two characters are not checked. The field passes validation if the first three characters are greater than 123.

Section 5  
Validation and Generation Descriptors

I4="6||5"

Validates a four-character integer field (I4) by checking the first character against the string constant 6 and the last character against the string constant 5. The second and third characters are not checked.

N5G"12345"

Generates (G) a five-character numeric field (N5) from the string constant 12345.

HEXADECIMAL STRING CONSTANT VALIDATION/GENERATION DESCRIPTOR [\]

Use the \ descriptor to validate against or generate from a string of hexadecimal values.

The syntax for the \ descriptor is:

`\xx[xx...]`

where xx is a two-digit hexadecimal code.

Usage Notes

- The \ descriptor can be used in the same places as the text string constant descriptor. It is most often used to embed control characters in a record (using the SET command) or to test for a specified control character (using an IF statement).
- Each pair of hexadecimal digits occupies one character space (one byte). For example, the hexadecimal string ODOA would occupy two columns in a field or buffer.
- If the hexadecimal code represents a character that cannot be displayed, it is replaced by a tilde (~) on the screen.
- The \ descriptor is used primarily in formats that manipulate asynchronous devices through the MSFP AS file type. See the LIfE-Works Format Language Manual, Volume 2 for information about MSFP and the AS file type.
- If a field is set to a hexadecimal value that is shorter than the field length, the hexadecimal value is left-justified within the field, and the rest of the field is filled with binary zeroes.

Examples

`SET F2=\ODOA`

Sets the hexadecimal values 0D (carriage return) and 0A (line feed) into field 2.

`IF F1=\04 THEN F6 ELSE F2`

If field 1 contains the hexadecimal value 04 (end of transmission), goes to program level 6; otherwise, goes to program level 2.









Section 6  
Screen Formatting Commands

INTRODUCTION

This section describes the format commands used to develop a screen display for the data entry operator. These commands move the cursor across the screen, place prompts and other text on the screen, erase characters, and so on. Table 6-1 describes the screen-formatting commands.

Table 6-1. Screen-Formatting Commands

Command	Description
Br,c	Blanks the screen from the current cursor position to a specified row and column.
Bw	Blanks the screen for a specified number of columns.
Er,c	Erases the screen from the current cursor position to a specified row and column.
Ew	Erases the screen for a specified number of columns.
EUA r,c	Erases unprotected screen characters from the current cursor position to a specified row and column.
EUA w	Erases unprotected screen characters for a specified number of columns.
P	Places a prompt on the screen, starting at the current cursor location.
Pr,c	Places a prompt on the screen, starting at a specified row and column.
T	Places a text string on the screen, starting at the current cursor position.
@r,c	Moves the cursor to a specified row and column.
@w	Moves the cursor a specified number of columns.



General Usage Notes

- At the beginning of a format, the cursor is positioned at row 1, column 1. As you write format code, you use the screen-formatting commands to move the cursor and place characters on the screen.

You must always move the cursor forward (to the right) and down. You cannot move the cursor backward to a previous column or up to a previous row.

- There are two basic techniques for creating screen displays. The first technique displays a prompt, moves the cursor to the position where the operator is to enter data, and waits for the entry. When the operator completes the entry, the next prompt appears and the cursor moves to the next entry position. This sequence continues until the end of the format.

For example:

```
P"NAME: "      /* DISPLAYS THE PROMPT NAME */
G20            /* DEFINES A 20-CHARACTER GENERAL ENTRY FIELD */
B2,1          /* MOVES THE CURSOR TO THE NEXT LINE */
P"ADDRESS: "   /* DISPLAYS THE PROMPT ADDRESS */
G30            /* DEFINES A 30-CHARACTER GENERAL ENTRY FIELD */
```

The second technique places all the prompts on the screen at once, then moves the cursor from field to field as the operator enters data. This technique lets you design a screen that resembles a paper source document. However, it is slightly more complex to code.

To use this technique, put all of the code for the prompts at the start of the format, and define the prompts using the Pr,c command. The Pr,c command places each prompt at the specified screen position, but does not move the cursor. You must then use the @r,c command to position the cursor at the first entry point and to reposition the cursor after each entry.

For example:

```
P2,1"NAME:"    /* DISPLAYS THE PROMPT NAME */
P3,1"ADDRESS:" /* DISPLAYS THE PROMPT ADDRESS */
@2,12         /* MOVES THE CURSOR TO ROW 2, COLUMN 12 */
G20           /* DEFINES A 20-CHARACTER GENERAL ENTRY FIELD */
@3,12         /* MOVES THE CURSOR TO ROW 3, COLUMN 12 */
G30           /* DEFINES A 30-CHARACTER GENERAL ENTRY FIELD */
```

- Screen formatting within a subroutine requires a specific coding technique. The cursor position at the start of a subroutine may vary, depending on when the subroutine is called. Therefore, you cannot use the screen-formatting commands that specify an absolute row and column position (r,c). Instead, use the commands that specify a number of columns (w). For example, use Bw rather than Br,c, Ew rather than Er,c, and so on.
- The following screen-formatting commands generate attributes (internal codes that distinguish prompts from data):
  - B (blank) commands. Blanks placed on the screen by a B command are treated as prompts.
  - P (prompt) commands. Characters placed on the screen by a P command are treated as prompts.
  - T (text) commands. Characters placed on the screen by a T command are treated as data.
  - @ (tab) commands. Characters in the path of an @ command are converted to prompts, regardless of their original attribute.

You can use attribute modifiers to alter the display characteristics of these elements. For example, you can display a title in reverse video or a prompt in high intensity. The specific syntax for using attribute modifiers with each element appears in this section; general information on attribute modifiers appears in Section 4, "Attributes."

If your system is configured for attribute spaces, you must allow for the spaces as you write your code. See Section 4 for more information.

- Characters placed on the screen by a P, T, or @ command can be assigned a field number, just as an entry field can. (See Section 2, "Field Types," for information on field numbers and entry fields.) The field number lets you refer to the corresponding element in a format statement. A field number can be any whole number from 1 to 999. You can assign up to 999 field numbers in a given format.
- In the syntax descriptions for the P, T, and @ commands, the literal part of the command (the part you type exactly as shown) is underlined for clarity.

BLANK TO ROW AND COLUMN COMMAND [Br,c]

Use the Br,c command to blank the screen from the current cursor position to a specified row and column on the screen.

The syntax for the Br,c command is:

B[(i)]r,c

where

i is an optional attribute modifier. i must be enclosed in parentheses.

r is the row number.

c is the column number. Column c is not blanked.

Usage Notes

- The Br,c command moves the cursor to row r, column c.
- Any prompts or data blanked by the Br,c command are removed from the screen and are not written to the disk.
- The operator cannot make any entry in an area blanked by the Br,c command. The format can place characters in a blanked area by using the SET S or Pr,c commands.
- Be sure to include the comma between the r and c values. If you don't, Life-Works assumes you are using the Bw command.
- See also "Blank to Column Command [Bw]" in this section.

Examples

B3,10

Blanks (B) the screen from the current cursor location up to but not including row 3 column 10.



BLANK COLUMNS COMMAND [Bw]

Use the Bw command to blank the screen for a specified number of columns, starting at the current cursor location.

The syntax for the Bw command is:

B[(i)]w

where

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the number of columns to be blanked.

Usage Notes

- The Bw command moves the cursor to the column immediately following the blanked columns.
- Any prompts or data blanked by the Bw command are removed from the screen and are not written to the disk.
- The operator cannot make any entry in an area blanked by the Bw command. The format can place characters in a blanked area by using the SET S command or the Pr,c command.
- See also "Blank to Row and Column Command [Br,c]" in this section.

Examples

B25

Blanks 25 columns on the screen (or blanks 26 columns if the system uses attribute spaces).



Section 6  
Screen-Formatting Commands

ERASE TO ROW AND COLUMN COMMAND [Er,c]

Use the Er,c command to erase the screen from the current cursor position to a specified row and column.

The syntax for the Er,c command is:

Er,c

where

r is the row number.

c is the column number. Column c is not erased.

Usage Notes

- The Er,c command does not move the cursor.
- The Er,c command is typically used at the beginning of a format to erase the screen of any leftover or unwanted data.
- The Er,c command works only in new records.
- Be sure to include the comma between the r and c values. If you don't, LiFE-Works assumes that you are using the Ew command.
- See also "Erase Columns Command [Ew]" in this section.

Examples

B2,1 P'DATE: ' E22,1

Blanks the screen to row 2, column 1, displays the prompt DATE:, and then erases the screen up to, but not including row 22, column 1.

E23,1

Erases the entire screen up to the message line. This command is often used immediately after the \$FORMAT statement to clear the screen for each new record. This command works for either an 80-character screen or a 96-character screen.

E25,1

Erases the entire screen including the message line and the status line. This command works for either an 80-character screen or a 96-character screen.

Section 6  
Screen-Formatting Commands

ERASE COLUMNS COMMAND [Ew]

Use the Ew command to erase a specified number of columns on the screen, starting at the current cursor position.

The syntax for the Ew command is:

Ew

where w is the number of columns to be erased.

Usage Notes

- The Ew command does not move the cursor.
- The Ew command is typically used at the beginning of a format to erase the screen of any leftover or unwanted data.
- The Ew command takes effect only in new records.
- See also "Erase to Row and Column Command [Er,c]" in this section.

Examples

E18

Erases the next 18 columns in the current record.

E1680

Erases 1680 characters, or 21 rows of 80 columns each. Used at the beginning of a record, this command erases all of an 80-character screen area except the last three rows.

ERASE UNPROTECTED COLUMNS TO ROW AND COLUMN COMMAND [EUAr,c]

Use the EUAr,c command to erase all unprotected display characters from the current cursor position to a specified row and column.

The syntax for the EUAr,c command is:

EUAr,c
--------

where

r is the row number.

c is the column number. Column c is not erased.

Usage Notes

- Characters placed on the screen by a prompt (P) command or a text (T) command are protected. If your system uses attribute spaces, the invisible attribute characters are also protected. All other characters are unprotected.
- The EUAr,c command does not move the cursor.
- See also "Erase Unprotected Columns Command [EUAw]" in this section.



ERASE UNPROTECTED COLUMNS COMMAND [EUAw]

Use the EUAw command to erase all unprotected display characters in a specified number of columns, starting from the current cursor position.

The syntax for the EUAw command is:

EUAw
------

where w is the number of columns to be erased.

Usage Notes

- Characters placed on the screen by a prompt (P) command or a text (T) command are protected. If your system uses attribute spaces, the invisible attribute characters are also protected. All other characters are unprotected.
- The EUAw command does not move the cursor.
- See also "Erase Unprotected Columns Command [EUAr,c]" in this section.

#### PROMPT DISPLAY COMMAND [P]

Use the P command to place a prompt on the screen, starting at the current cursor position.

The syntax for the P command is:

`[f]P[(i)]"text"`

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

"text" is the text of the prompt. text can be up to 180 characters long and must be enclosed in quotation marks.

#### Usage Notes

- Use the P command when you want to place prompts one at a time on the display screen, then wait for the operator's entry. When you use P to display a prompt, the cursor moves to the screen position immediately following the last character of the prompt.
- If your system is configured for attribute spaces, the P command places an attribute space before each prompt. To make a prompt start in the first screen position (row 1, column 1), you must use the ATR command to move the attribute from row 1, column 1. See Section 4, "Attributes," for details.
- You can use the SET screen command (SET S) to change a prompt. The SET S command overwrites an existing prompt.
- You can use the Pw command interchangeably with the P command. The syntax for the Pw command is:

`[f]P[(i)]w,text`

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the number of characters in the prompt (1-180).

Section 6  
Screen-Formatting Commands

text is the text of the prompt.

Some format programs transferred from a Series 4000/5000 system may use Pw.

Examples

P"NAME: " E23,1

Places the prompt NAME: at the current cursor position and erases the screen up to, but not including, row 23 column 1.

B5,1 2P"TOTAL CHARGES:"

Blanks the screen from the current cursor location up to, but not including, row 5 column 1, identifies the prompt as field 2, then places the prompt TOTAL CHARGES: on the screen.

PROMPT DISPLAY AT ROW AND COLUMN COMMAND [Pr,c]

Use the Pr,c command to place a prompt on the screen, starting at a specified row and column.

The syntax for the Pr,c command is:

[f]P[(i)]r,c"text"

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

r is the specified row.

c is the specified column.

"text" is the prompt. Each prompt can be up to 180 characters long.

Usage Notes

- The Pr,c command is useful if you want the format to display all prompts before the operator enters any data. When you use Pr,c, the cursor does not move with the prompts; it remains at the current location on the screen.
- If your system is configured for attribute spaces, the Pr,c command places an attribute space before each prompt.
- You can use the SET screen command (SET S) to overwrite an existing prompt.
- See also the "Prompt Display Command [P]" in this section.

Examples

P2,5"NAME:"

Places the prompt "NAME:" on the screen, starting in row 2 column 5.



Section 6  
Screen-Formatting Commands

4P5,1"TOTAL CHARGES:"

Identifies the prompt TOTAL CHARGES: as field 4 and places it on the screen, starting in row 5 column 1.

### GENERATE TEXT COMMAND [T]

Use the T command to place a text string on the screen, starting at the current cursor location.

The syntax for the T command is:

[f]T[(i)]"text"

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

text is the text string. The string can be up to 180 characters long and must be enclosed in quotation marks.

### Usage Notes

- The T command functions much like the G modifier (automatic generate). It does not require any keystrokes from the operator, and all characters displayed on the screen become a part of the disk record. The T command does not require ASD to be on.
- The T command generates text only in a new record or in key verify mode (assuming the verify feature is on).
- After the text is placed on the screen, the cursor moves to the screen position immediately following the last column of text. If necessary, the data entry operator can backspace into the generated text and type over it.
- You can use the Tw command interchangeably with the T command. The syntax for the Tw command is:

[f]T[(i)]w,text

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the number of characters in the text string (1-180).

text is the text string.

Section 6  
Screen-Formatting Commands

Some format programs transferred from a Series 4000/5000 system may use Tw.

Examples

3T'ACE OFFICE SUPPLIES, DOWNTOWN STORE '

Generates the text ACE OFFICE SUPPLIES, DOWNTOWN STORE (with a space after STORE), starting at the current cursor position. The generated text is identified as field 3 (3T).

#### TAB TO ROW AND COLUMN COMMAND [@r,c]

Use the @r,c command to move the cursor from the current location to a specified row and column. The @r,c command moves the cursor over characters and attributes without erasing them; instead, it turns all characters in its path into prompts.

The syntax for the @r,c command is:

[f]@[(i)]r,c

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

r is the specified row number.

c is the specified column number.

#### Usage Notes

- The @ commands are called the "tab" commands.
- Use the @r,c command to convert text or data remaining on the screen from a previous format into prompts, or to tab over prompts you have already placed on the screen.
- If you are moving the cursor over a blank screen area, the B commands are faster than the @ commands.
- See also "Tab to Column Command [@w]" in this section.

#### Examples

P3,10'DEPARTMENT: '  
@3,22 G4EF

Places the prompt DEPARTMENT: on the screen starting in row 3, column 10. The @3,22 G4EF command moves the cursor to row 3, column 22, and defines the data entry field for the DEPARTMENT: prompt as a general field (G4) that must be entered (E) and must be filled (F).



#### TAB TO COLUMN COMMAND [@w]

Use the @w command to move the cursor a specified number of columns, starting from the current cursor position. The @w command moves the cursor over characters and attributes without erasing them; instead, it turns all characters in its path into prompts.

The syntax for the @w command is:

[f]@[(i)]w

where

f is an optional field number.

i is an optional attribute modifier. i must be enclosed in parentheses.

w is the number of columns.

#### Usage Notes

- The @ commands are called the "tab" commands.
- Use the @w command to convert text or data remaining on the screen from a previous format into prompts, or to tab over prompts you have already placed on the screen.
- If you are moving the cursor over a blank screen area, the B commands are faster than the @ commands.
- See also "Tab to Row and Column Command [@r,c]" in this section.





## Section 7 Other Commands

### INTRODUCTION

This section describes the commands listed in Table 7-1. These commands perform various system actions such as turning features on and off, calling subroutines, and transferring control to other jobs and batches.

For descriptions of additional LiFE-Works commands, see the following sections:

- Section 6, "Screen Formatting Commands," describes commands used for creating a screen display.
- Section 8, "Arithmetic and Logic," describes commands used in conditional logic statements.
- Section 10, "Index Sets," describes commands used with index sets.

### General Usage Notes

- Some LiFE-Works commands are compiler directives. A compiler directive takes effect when the format is compiled rather than when it is executed.

Avoid using a compiler directive in an IF or SPLIT statement. A directive in a conditional statement affects any format statements that follow the directive, regardless of which path in the conditional statement is actually taken at runtime.



Table 7-1. Commands

Command	Description
ASD	Turns the automatic skip-duplicate feature (ASD) on.
ASD#	Turns the automatic skip-duplicate feature (ASD) off.
BEEP	Makes the terminal beep (audible alarm).
BLANKB	Blanks the work buffer.
BRANCH	Transfers control to a specified job and batch, and begins entry mode.
C	Generates a check digit.
CALL	Calls a specified subroutine.
COMMENT...; or /*comment*/	Lets you add comments to your code. A comment is an explanatory statement that does not execute.
CORR	Lets the data entry operator change a record without pressing the CORR key.
DISPLAY	Causes system-generated characters to be displayed on the screen (reverses the effect of the DISPLAY# command).
DISPLAY#	Prevents system-generated characters from being displayed on the screen.
EDIT	Assigns a value to an accumulator, work buffer, field, record buffer, index set, or screen area, and edits the value to conform to a specified format (picture).
END	Indicates the end of a main program that is followed by one or more subroutines.
ENDSUB	Indicates the end of a subroutine.
EXEC	Executes a specified C function.
F	Starts a new program level.
FDR	Forces a record to be written to the disk.
GETBUF	Allocates a blank work buffer to the terminal.
INCLUDE	Opens a specified job and batch, and compiles the source code in that job and batch along with the main source code.

Table 7-1. Commands (Continued)

Command	Description
LOW	Turns the lowercase feature on.
LOW#	Turns the lowercase feature off.
MODE	Enables the EXIT key while a format program is running (reverses the effect of the MODE# command).
MODE#	Disables the EXIT key while a format program is running.
NDR	Prevents the data on the screen from being written to disk.
PRINTS	Prints the screen.
RELBUF	Releases the current terminal's work buffer.
SET	Assigns a value to an accumulator, work buffer, field, record buffer, index set, or screen area.
SLEEP	Suspends execution of the format for a specified number of seconds.
SUBROUTINE	Indicates the beginning of a subroutine and assigns a name to the subroutine.
SUPERM	Transfers to a specified job and batch, and executes the batch in mode T-M.
SUPERS	Transfers to a specified job and batch, and executes the batch in mode T-S.
USE BRANCH	Executes a specified statement in a SPLIT statement. USE BRANCH is described in Section 8, "Arithmetic and Logic."
V	Turns the verify feature on. When verify is on, data can be checked in key verify mode.
V#	Turns the verify feature off. When verify is off, data cannot be key verified.
VALID	Flags records for sight verify.
Z	Clears a specified accumulator to zero.

Section 7  
Other Commands

AUTO SKIP DUPLICATE COMMAND [ASD]

Use the ASD command to turn the automatic skip-duplicate feature (ASD) on.

The syntax for the ASD command is:

ASD
-----

Usage Notes

- The automatic skip-duplicate feature allows automatic skipping, duplicating and generating of data as directed by the format program.
- The ASD feature is off at the beginning of each job; however, once it is turned on, it remains on until it is turned off by the ASD# command, or until the operator presses the ASD key.
- See also "Auto Skip Duplicate Off Command [ASD#]" in this section.

AUTO SKIP DUPLICATE OFF COMMAND [ASD#]

Use the ASD# command to turn the automatic skip-duplicate feature (ASD) off.

The syntax for the ASD command is:

ASD#
------

Usage Notes

- See also "Auto Skip Duplicate Command [ASD]" in this section.



AUDIBLE ALARM COMMAND [BEEP]

Use the BEEP command to make the terminal beep (sound an audible alarm).

The syntax for the BEEP command is:

BEEP
------

Usage Notes

- BEEP is commonly used to indicate an error to the operator.
- BEEP does not execute in pass 1 or during verify reconstruct. (Pass 1 is described in Appendix A, "Record Processing.")
- BEEP has no effect if your system is not configured for the audible alarm.

BLANK WORK BUFFER COMMAND [BLANKB]

Use the BLANKB command to clear the work buffer of previous information before entering new data.

The syntax for the BLANKB command is:

BLANKB
--------

Usage Notes

- Each work buffer is blank at initialization. Use the BLANKB command to clear and blank-fill the buffer after the format has used the buffer for storage.
- For general information about work buffers, see "Get Work Buffer Command [GETBUF]" in this section.
- BLANKB applies only to the work buffer, not to MSFP record buffers. For commands used with MSFP buffers, see the LIfE-Works Format Language Manual, Volume 2.

### BRANCH COMMAND [BRANCH]

The BRANCH command transfers control from the current job and batch to a specified job and batch and begins entry mode.

The syntax for the BRANCH command is:

BRANCH=reference

where reference is either the name of the target job and batch, or a location where the target job and batch name can be found, as follows:

"job,batch" identifies the target job and batch. The string must be enclosed in quotation marks.

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c and extending for l columns.

Ff refers to field f of the current format.

LFf refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c and extending for l columns.

RS(n,l) refers to a relative screen area within a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to a screen area beginning in row r, column c, and extending for l columns.

X(id,c,l) refers to index set id, beginning in column c and extending for l columns.

.symbol. refers to a name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

### Usage Notes

- Before transferring control, the BRANCH command releases the current record, writes it to disk, and closes the batch. (You can use the NDR command to prevent the record from being written to disk.)
- Work buffers are not released or blanked when the BRANCH command is executed, but accumulators are cleared to zero.

- The BRANCH command is usually placed near the end of the format.
- The data entry operator must enter at least one character in the record before BRANCH will execute (unless you use the NDR command). No data fields should follow the BRANCH command unless BRANCH is part of an IF or SPLIT statement.
- The BRANCH command executes only in new records or during key verify.
- The BRANCH command does not return to the calling point in the original format; it remains in the new format. BRANCH cannot be used in subroutines.
- If BRANCH specifies a password-protected job, the data entry operator is asked to enter the password before continuing.
- If BRANCH specifies a nonexistent target batch, the batch is created when BRANCH executes.

#### Examples

```
BRANCH="SUMMARY,1"
```

Branches to the job and batch SUMMARY,1.

```
BRANCH=F2
```

Branches to the job and batch specified in field 2.

```
BRANCH=.NEWJOB.
```

Branches to the job and batch specified by the symbol NEWJOB.



### GENERATE CHECK DIGIT COMMAND [C]

The C command generates a check digit from the previous field or fields and displays it on the screen.

The syntax for the C command is:

[f]Cw,m

where

f is an optional field number.

w is the length of the field the check digit is generated from, plus one for the check digit.

m is the modulus (7, 10, or 11).

### Usage Notes

- Check-digit generation does not require ASD to be on.
- If the operator releases the record before the C command is executed, a blank is generated in place of a check digit.
- The w columns can span several fields, including automatic duplicate fields.

However, if your system is configured for attribute spaces, no attribute spaces are allowed between these fields. You must use the (-) modifier to suppress the attribute. See Section 4, "Attributes," for details.

- The letters A through Z can be included, but only the digit part of the letter when represented as a Hollerith card code is used to compute the check digit. See Appendix B for more details on this.
- See also Appendix B, "Check Digit Calculation."
- See also "Check Digit Modifier [C]" in Section 3, "Field Modifiers."

### Examples

I8 C9,11

Generates a check digit using modulus 11 and places it after the last digit of the previous field (I8). The length of the base number is 8; the length of the entire number is 9.

CALL SUBROUTINE COMMAND [CALL]

Use the CALL command to call a specified subroutine.

The syntax for the CALL command is:

```
CALL "name"
```

where name identifies the subroutine to be called. name must be enclosed in quotation marks.

Usage Notes

- Place the CALL command at the point in the code where you want the called subroutine to execute. CALL can appear within the main format program or within a subroutine.
- See also "Define Subroutine Command [SUBROUTINE]" in this section.

Examples

```
IF F1!0 THEN CALL "OVERDUE" ELSE NULL
```

If field 1 is not equal to zero, call the subroutine OVERDUE; otherwise, take no action.

Section 7  
Other Commands

COMMENT COMMAND [COMMENT OR /\*comment\*/]

Use the COMMENT command or the /\* and \*/ delimiters to add comments to your format code.

The syntax for the comment command is:

```
COMMENT comment;
```

or

```
/*comment*/
```

where comment is the text of the comment.

Usage Notes

- The format compiler ignores all characters between COMMENT and the semicolon. When you use the COMMENT command, you cannot use a semicolon in the text of the comment. The word COMMENT must be separated from the comment by at least one space.
- The format compiler ignores all characters between the /\* and \*/ delimiters. You can optionally place one or more separating blanks between the delimiters and the text of the comment.

Examples

```
COMMENT READ EMPLOYEE FILE;
```

```
/* READ EMPLOYEE FILE */
```

Two different ways to add the comment READ EMPLOYEE FILE to the code.

CORRECTION COMMAND [CORR]

The CORR command automatically opens an existing record for correction in entry mode or find mode. When you use CORR in a format, the operator does not have to press the CORR key before modifying, inserting, or deleting a record.

The syntax for the CORR command is:

CORR
------

Usage Notes

- Use the CORR command with caution; it increases the chance that an operator will alter or delete data inadvertently.
- Place the CORR command before the first format statement that specifies entry fields or screen positions.
- The CORR command affects all existing records in the current format. It has no effect on new records or in key verify mode.



RESUME CHARACTER DISPLAY COMMAND [DISPLAY]

The DISPLAY command causes a return to normal screen display of system-generated characters.

The syntax for the DISPLAY command is:

DISPLAY
---------

Usage Notes

- DISPLAY reverses the effect of the DISPLAY# command, which prevents system-generated characters from being displayed. See "Inhibit Character Display Command [DISPLAY#]" in this section for more information.

INHIBIT CHARACTER DISPLAY COMMAND [DISPLAY#]

The DISPLAY# command prevents system-generated characters from being displayed on the screen.

The syntax for the DISPLAY# command is:

DISPLAY#
----------

Usage Notes

- DISPLAY# inhibits characters generated by prompt fields (P), text fields (T), the SET command (SET S), and entry fields generated by the G modifier.

DISPLAY# has no effect on operator input. The operator's keystrokes are always displayed on the screen.

- The DISPLAY command causes a return to normal character display. See "Resume Character Display [DISPLAY]" in this section.
- Displaying information on the screen slows down format execution. You can use DISPLAY# and DISPLAY to bracket sections of code containing screen displays that are not always needed. For example, you might want certain information on the screen while you are debugging a format, but not when the format is actually in use. You may also be able to speed up some programs transferred from Series 4000/5000 VISION by inhibiting unneeded screen displays.

#### EDIT NUMERIC FIELD COMMAND [EDIT]

The EDIT command assigns a numeric value to a location, such as a field or buffer. EDIT lets you specify a "picture" that determines the format of the assigned value.

The syntax for the EDIT command is:

EDIT area=value USING picture

where

area is a location into which a value is to be set. area can be any of the following:

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c and extending for l columns.

Ff refers to field f of the current format.

Lff refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c and extending for l columns.

RS(n,l) refers to a relative screen area within a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to the screen area beginning in row r, column c, and extending for l columns.

X(id,c,l) refers to index set id, beginning in column c and extending for l columns.

.symbol. refers to a symbolic name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

value is either the value to be assigned or a location where the value can be found. value can be any of the following:

"string" is a character string, such as "1949." The string must be enclosed in quotation marks.

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c and extending for l columns.



Ff refers to field f of the current format.

LFf refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c and extending for l columns.

RS(n,l) refers to a relative screen area within a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to the screen area beginning in row r, column c, and extending for l columns.

X(id,c,l) refers to index set id, beginning in column c and extending for l columns.

.symbol. refers to a symbolic name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

picture is a series of symbols that determines the format of value. Table 7-2 describes the EDIT picture symbols.

Table 7-2. EDIT Picture Symbols

Symbol	Explanation
9	Represents a numeric character. For example, the picture 9999 represents a four-digit number.
Z	Represents a numeric character and indicates that leading zeroes are to be suppressed. For example, the picture ZZZZ represents a four-digit number; any leading zeroes in the number will be dropped.
* (asterisk)	Represents a numeric character and indicates that leading zeroes are to be replaced by asterisks. For example, the picture **** represents a four-digit number; asterisks will be substituted for any leading zeroes.
. (period)	Represents a decimal point. For example, the picture 999.9 represents a numeric value with one decimal place.  If specified, the decimal point always appears unless the Z zero suppression character is used.



Table 7-2. EDIT Picture Symbols (Continued)

Symbol	Explanation
, (comma) 0 (zero) B (blank)	<p>The comma, zero, and blank (represented by a B) are literals to be inserted into the output in the position shown. For example, the picture 9,999.00 represents a numeric value with a comma separating thousands from hundreds and with two zeroes following the decimal point.</p> <p>The comma, zero, and blank are suppressed if they appear within leading zeroes and if zero suppression is being used (Z or *).</p>
+ (plus) - (minus)	<p>Represent a plus or minus sign to be inserted into the output in the position shown. The plus or minus symbol must be either the first or last character in the picture. For example, the picture +999 represents a three-digit value preceded by a plus sign if the value is positive, or by a minus sign if the value is negative. The picture -999 represents a three-digit value preceded by a blank if the value is positive, or by a minus sign if the value is negative.</p> <p>The plus and minus symbols can also be used to designate zero suppression and to float the symbol next to the significant digit. For example, if the picture is ----9 and the data is -0443, the resulting value is -443.</p>
CR DB	<p>Represent a CR or DB symbol to be inserted into the output, immediately following the numeric value. The CR or DB symbol must be the rightmost symbol in the picture. If the data is negative, the symbol used in the picture (either CR or DB) is inserted. If the data is positive, two blanks are inserted instead of the CR or DB.</p>
\$	<p>Represents a dollar sign in the output. Can be used instead of a zero suppression symbol to suppress leading zeroes and float a dollar sign next to the first significant digit. For example, the picture \$\$\$\$ represents a three-digit value (not four) with a leading dollar sign. The leftmost \$ symbol represents the leading dollar sign; the remaining \$ symbols represent numeric characters, with leading zeroes suppressed.</p>
(picture)	<p>Enclosing the entire picture in parentheses causes the output value to be enclosed in parentheses if it is negative, or to be enclosed in blanks if it is positive.</p>

### Usage Notes

- The EDIT command has the same general function as the SET command, but EDIT lets you alter the format of the assigned value.
- An EDIT picture can be up to 18 characters (symbols) long.
- If there are differences in the lengths of the source value and the EDIT picture, or in the lengths of the picture and the destination area, LIfE-Works resolves the differences by truncating or padding. LIfE-Works uses the following rules:
  - Any truncating or padding is first done between the source value and the picture, then between the picture and the destination area.
  - If the length of value is less than the number of numeric positions in the picture, LIfE-Works pads the left side of value with zeroes before the edit.
  - If the length of value is greater than the number of numeric positions in the picture, LIfE-Works truncates value on the left before the edit.
  - If the length of the picture is less than the length of area, LIfE-Works pads the right side of the output value with blanks.
  - If the length of the picture is greater than the length of area, LIfE-Works truncates the rightmost characters of the output value.
- To use the European convention in which a comma serves as a decimal point, use the compiler directive DECIMAL POINT IS COMMA. The roles of the comma and the period are reversed for statements appearing after this directive. To return to the American convention, use the directive DECIMAL POINT IS PERIOD.

Either convention can be selected as the default. These directives should not be used inside an IF or SPLIT statement.

### Examples

#### The 9 symbol:

Picture	Data	Result
99999	00001	00001
99999	12345	12345
99999	45	00045

Section 7  
Other Commands

The zero suppression symbols (Z and \*):

Picture	Data	Result
ZZZZZ	00365	365
ZZZZZ	00000	
ZZZ.ZZ	00001	.01
*****	00365	**365
*****	00000	*****
***.**	00001	***.01

The decimal point symbol (.):

Picture	Data	Result
9,999.99	623417	6,234.17
ZZZ.ZZ	00.21	.21

The comma, zero, and blank symbols:

Picture	Data	Result
99,999	45206	45,206
000.99	01	000.01
99B999	56789	56 789
**,***	00032	****32
ZZ,ZZZ	00032	32

The plus and minus symbols:

Picture	Data	Result
+9999	5421	+5421
+9999	-5421	-5421
9999-	5421	5421
9999-	-5421	5421-
+++99	3497	+3497
+++99	-0043	-43

The CR and DB symbols:

Picture	Data	Result
9999CR	5734	5734
9999CR	-5734	5734CR
9999DB	5734	5734
9999DB	-5734	5734DB

The \$ symbol:

Picture	Data	Result
\$\$,\$\$\$ .99	000127	\$1.27
\$\$,\$\$\$ .99	927456	\$9,274.56

The () symbols:

Picture	Data	Result
(9999)	3211	3211
(9999)	-3211	(3211)



END SUBROUTINE COMMAND [ENDSUB]

Use the ENDSUB command to indicate the end of the current subroutine.

The syntax for the ENDSUB command is:

ENDSUB
--------

Usage Notes

- All subroutines must follow the main format code. The general syntax is:

```
$FORMAT 001 $SUB      /* Main format code */
.
CALL "ONE"             /* Call first subroutine */
.
CALL "TWO"             /* Call second subroutine */
.
.
END                   /* End of main format */
SUBROUTINE "ONE"       /* Start of first subroutine */
.
.
ENDSUB                /* End of first subroutine */
SUBROUTINE "TWO"       /* Start of second subroutine */
.
.
ENDSUB                /* End of second subroutine */
```

- If the format contains more than one subroutine, ENDSUB should be followed by the next SUBROUTINE statement. ENDSUB is usually the last statement in a format.

### EXECUTE C FUNCTION COMMAND [EXEC]

Use the EXEC command to execute a function that has been coded in the C programming language. The function must exist when the format is run.

The syntax for the EXEC command is:

EXEC(i[,argument,argument...])

where

i is the function index (1-999999). i can be a numeric constant or one of the following validation/generation descriptors:

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c and extending for l columns.

Ff refers to field f of the current format.

LFf refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c and extending for l columns.

RS(n,l) refers to a relative screen area within a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to the screen area beginning in row r, column c, and extending for l columns.

X(id,c,l) refers to index set id, beginning in column c and extending for l columns.

.symbol. refers to a symbolic name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

argument is an optional argument that is passed to the function as a character string. You can send up to 49 arguments. argument can be any of the following validation/generation descriptors:

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c and extending for l columns.

Ff refers to field f of the current format.

LFf refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c and extending for l columns.

RS(n,l) refers to a relative screen area within a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to the screen area beginning in row r, column c, and extending for l columns.

X(id,c,l) refers to index set id, beginning in column c and extending for l columns.

.symbol. refers to a symbolic name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

#### Usage Notes

- EXEC executes in both pass 1 and pass 2. (See Appendix A, "Record Processing," for information about pass 1 and pass 2.)
- Be aware that if the C function changes the value of a screen area, the change is not automatically displayed. It is strongly recommended that you make all changes in an offscreen area, such as an accumulator or buffer.
- To use the EXEC command, the \$W/cd directory must be installed on your system. See the LIfE-Works System Release Guide for additional information.
- For a detailed example, see the source module cd.exec.c in the \$W/cd directory.
- See also "Execute C Function Modifier [X]" in Section 3, "Field Modifiers."



PROGRAM LEVEL COMMAND [F]

The F command releases the current record and selects a new program level.

The syntax for the F command is:

Fn

where n is the program level to be selected. n can be a number from 1 to 15.

Usage Notes

- The number of each program level is determined by the order in which the format is assigned to the job in the \$DEFJOB command. In other words, the first format assigned to the job becomes program level 1 for that job, the second format assigned to the job becomes program level 2 for that job, and so on. Each job can have up to 15 program levels.
- If no F command is given, the current format is run again.
- The data entry operator can select a program level manually by pressing the SELECT key, then entering a two-digit number from 01 to 15.
- The data entry operator must enter at least one data character in the record before the F command will execute (unless the NDR command is used).
- There should be no data fields following the F command except as part of an IF or SPLIT statement.
- The F command executes only in new records.

Examples

F3

Selects program level 3.



FORCE DISK RECORD COMMAND [FDR]

The FDR command forces a record to be written to the disk in entry, find, or key verify modes.

The syntax for the FDR command is:

FDR
-----

Usage Notes

- If FDR is not used, existing records are not written to disk in entry, find, or key verify mode unless the record has been opened for correction.
- FDR is often used to send data to a host after the data has been key verified. See Section 12, "Format Programming for Interactive Communications," for an example.

#### GET WORK BUFFER COMMAND [GETBUF]

The GETBUF command allocates a blank work buffer for the current terminal.

The syntax for the GETBUF command is:

GETBUF[(SIZE=s)]

where s specifies the size of the work buffer. s can be any value between 1 and 2000. SIZE=s must be enclosed in parentheses. If you do not specify a work buffer size, the default size is 192 characters.

#### Usage Notes

- Do not allocate a larger buffer than you need; unnecessarily large buffers waste system resources.
- You can use the B validation/generation descriptor to validate or generate from the current terminal's work buffer. B is described in Section 5, "Validation/Generation Descriptors."
- GETBUF can be executed any number of times in any pass, but only the first execution assigns the work buffer to the terminal. This means each terminal has only one work buffer.
- The GETBUF command blanks the buffer initially, but you can use the BLANKB command to blank the buffer after it has been used. BLANKB is described in this section.
- The work buffer is not released or blanked when a BRANCH command is executed.
- Use the BUFFER conditional indicator to determine if a work buffer is currently allocated. BUFFER is described in Section 8, "Arithmetic and Logic."

#### Examples

GETBUF(SIZE=300)

Allocates a work buffer with a size of 300 characters.

INCLUDE JOB AND BATCH DIRECTIVE [INCLUDE]

The INCLUDE directive, when placed within the main body of a format program, tells the format compiler to open a specified job and batch and to compile the source code in that job and batch along with the main source code.

The syntax for the INCLUDE command is:

INCLUDE "job,batch"

where

"job,batch" specifies the job and batch to be included with the main source code for compiling. The string must be enclosed in quotation marks.

Usage Notes

- The format compiler compiles the code in the specified job and batch; when it reaches the end of the batch, it closes the batch and resumes compilation of the main source code with the statement following the INCLUDE command.
- You can't split format statements between the included batch and the main batch. In other words, don't start a prompt or command in the included batch and expect it to continue in the main batch.
- INCLUDES can be nested. However, a given job and batch cannot include itself.

### LOWERCASE ON COMMAND [LOW]

The LOW command turns on the lowercase feature to allow upper- and lowercase entries in the a and g fields.

The syntax for the LOW command is:

LOW
-----

### Usage Notes

- During data entry, if the lowercase feature is off, the a and g fields are the same as the A and G fields: that is, lowercase entries are allowed but they are forced to uppercase on the display screen.
- The lowercase feature can be controlled from the keyboard as well as through format code. The operator can turn the lowercase feature on or off by pressing the CASE key, and the format can turn the lowercase feature on or off by executing LOW or LOW#. In other words, the operator can override the LOW/LOW# commands by pressing CASE. Note that when the operator exits a batch, the system returns to whatever state it was in (lowercase on or lowercase off) before the batch was entered.
- See also "Lowercase Off Command [LOW#]" in this section.



Section 7  
Other Commands

LOWERCASE OFF COMMAND [LOW#]

The LOW# command turns off the lowercase feature and does not allow lowercase entries in the a and g fields.

The syntax for the LOW# command is:

LOW#
------

Usage Notes

- If the lowercase feature is off, the a and g fields are the same as the A and G fields: that is, lowercase entries are allowed but they are forced to uppercase.
- The lowercase feature can be controlled from the keyboard as well as through format code. The operator can turn the lowercase feature on or off by pressing the CASE key, and the format can turn the lowercase feature on or off by executing LOW or LOW#. In other words, the operator can override the LOW/LOW# commands by pressing CASE. Note that when the operator exits a batch, the system returns to whatever state it was in (lowercase on or lowercase off) before the batch was entered.
- See also "Lowercase On Command [LOW]" in this section.

ENABLE EXIT KEY COMMAND [MODE]

The MODE command returns the EXIT key to its normal mode-selection function. MODE reverses the effect of the MODE# command, which disables the EXIT key while a format is running.

The syntax for the MODE command is:

MODE
------

Usage Notes

- If the EXIT key is disabled, it remains disabled until a MODE command is executed in the current format or another format.
- See also "Disable EXIT Key Command [MODE#]" in this section.

DISABLE EXIT KEY COMMAND [MODE#]

The MODE# command disables the EXIT key while a format is running. MODE# prevents the operator from accidentally terminating a format by pressing EXIT.

The syntax for the MODE# command is:

MODE#
-------

Usage Notes

- If the EXIT key is disabled, it remains disabled until a MODE command is executed in the current format or another format.
- An operator can be signed off from another terminal (mode X-K) even when MODE# is in effect.
- The MODE command returns the EXIT key to normal operation within a format. See "Enable EXIT Key Command [MODE]" in this section.

NO DISK RECORD COMMAND [NDR]

The NDR command prevents the creation of a disk record at the end of format execution.

The syntax for the NDR command is:

NDR
-----

Usage Notes

- The NDR command is useful for manipulating data, such as index sets, when you don't want any data written to the disk.
- The NDR command is usually placed at the beginning of a format.
- If the format code encounters the NDR command in an existing record, the record is bypassed and it is not written to the disk.
- NDR does not operate during verify reconstruct.
- If an NDR command is not in an IF or SPLIT statement, there is no restriction on the number of characters that can be entered into the record. However, if zero characters are entered, or more than 1500 characters are entered, you cannot use the \$SIZE command in the \$FORMAT command statement.
- Once the NDR command is encountered, the format compiler does not check to see if the field lengths in a THEN/ELSE path of an IF statement match, or if the field lengths in the branches of a SPLIT statement match. However, the IF and SPLIT statements must create the same amount of space on the screen. This means that you might have to "pad" a field in the THEN path to match the corresponding field in the ELSE path.



Section 7  
Other Commands

NO BACKSPACING COMMAND [NOBACK]

Use the NOBACK command to prevent the data entry operator from backspacing in a record.

The syntax for the NOBACK command is:

NOBACK
--------

Usage Notes

- If the operator presses LEFT ARROW, SHIFT LEFT ARROW, or HOME after NOBACK is executed, the flashing message BACKSPACE IS INHIBITED appears, and entry is inhibited. The operator must press RESET to continue.
- Use NOBACK in situations where changes to a previously entered field might cause problems (for example, in index set updates or MSFP transactions). See Appendix A, "Record Processing," for information about what the system does and does not do when the operator backs up in a record.

NO RECORD/DOCUMENT UP COMMAND [NOUP]

Use the NOUP command to prevent the data entry operator from backing up to a previous record or document.

The syntax for the NOUP command is:

NOUP
------

Usage Notes

- If the operator presses UP ARROW, CTRL UP ARROW, or CTRL HOME after NOUP is executed, the flashing message INHIBITED KEY appears. The operator must press RESET to continue.
- Once NOUP executes, it stays in effect for the rest of the format.
- Use NOUP in situations where changes to a previously stored record might cause problems (for example, in index set updates or MSFP transactions). See Appendix A, "Record Processing," for information about what the system does and does not do when the operator backs up in a batch.

SCREEN PRINT COMMAND [PRINTS]

The PRINTS command prints a copy of the screen currently displayed. PRINTS works the same way as the data entry operator's screen print key.

The syntax for the PRINTS command is:

PRINTS
--------

Usage Notes

- The PRINTS command works only if screen printing is turned on for the terminal running the format. Screen printing can be turned on or off for each terminal through the \$SETPRINT supervisory command. See the LIFE-Works Supervisor's Manual for details.

- When PRINTS executes, the cursor stops flashing while the screen print is being queued. The message

Queuing request to print system

displays on the message line. When the screen print is queued, the cursor starts flashing and the format continues with the execution of the program.

- The PRINTS command executes only in new records and in existing records opened for correction. It does not execute in pass 1 or in key verify mode when the verify feature is not on. (Pass 1 is described in Appendix A, Record Processing.)

RELEASE WORK BUFFER COMMAND [RELBUF]

RELBUF releases the work buffer allocated to the current terminal.

The syntax for the RELBUF command is:

RELBUF
--------

Usage Notes

- RELBUF has no effect if no work buffer was allocated.
- The work buffer is automatically released when the operator exits the batch.
- See also "Get Work Buffer Command [GETBUF]" in this section.



### SET COMMAND [SET]

The SET command places a value in a specified location, such as a field, accumulator, or screen area.

The syntax for the SET command is:

SET area=value

where

area is a location into which a value is to be set. area can be any of the following:

An refers to accumulator n.

B(c,l) refers to a field in the work buffer, starting in column c and extending for l columns.

Ff refers to field f in the current record. The FDf and FPf forms of the F command are also valid.

LFf refers to local field f. (A local field is a field defined within a subroutine.) The LFDf and LFPf forms of the LF command are also valid.

Rfn(c,l) refers to a field in the record buffer for MSFP file fn, starting in column c and extending for l columns.

RS(n,l) refers to a relative screen area in a subroutine, starting n columns from the initial cursor position and extending for l columns. The RSD(n,l) and RSP(n,l) forms of the RS command are also valid.

S(r,c,l) refers to a screen area starting in row r, column c, and extending for l columns. The SD(r,c,l) and SP(r,c,l) forms of the S command are also valid.

X(id,c,l) refers to a field in index set id, starting in column c and extending for l columns.

.symbol. refers to a symbolic name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

value is the value to be set into the specified area. value can be any of the following:

- A numeric constant.
- A character string. The string must be enclosed in quotation marks.

- An arithmetic expression.
- A system keyword.
- A location where the value is to be found, as follows:

An refers to accumulator n.

B(c,l) refers to a field in the work buffer, starting in column c for a length of l characters.

Ff refers to field f in the current record.

LFf refers to local field f. (A local field is a field defined within a subroutine.)

Rfn(c,l) refers to a field in the record buffer for MSFP file fn, starting in column c for a length of l characters.

RS(n,l) refers to a relative screen area in a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to a screen area starting in row r, column c, and extending for l columns.

X(id,c,l) refers to a field in index set id, starting in column c for a length of l columns.

.symbol. refers to a symbolic name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

#### Usage Notes

- You can write SET statements that have nonmatching lengths--that is, statements in which the length of the left side (area) does not match the length of the right side (value) nor the length of accumulator 0. (LIFE-Works uses accumulator 0 for working storage while evaluating certain kinds of SET statements.) If you use nonmatching lengths, you need to know how LIFE-Works handles them. See "Nonmatching Lengths" in this section for details.
- You cannot use a numeric constant or an arithmetic expression in a SET statement where the left side length (area) is greater than the accumulator length.
- On systems that are not configured for attribute spaces, setting a screen area with the S(r,c,l) or RS(n,l) descriptors causes all characters in the specified screen area to be treated as data. On systems that are configured for attribute spaces, the attribute that precedes each field in the specified area determines whether the characters are treated as data or prompts.

Section 7  
Other Commands

In either case, you can use the D modifier (SD, RSD) on the left side of a SET statement to ensure that the resulting value is treated as data, or use the P modifier (SP, RSP) to ensure that the resulting value is treated as a prompt.

Examples

```
SET A1=S(1,1,12)
```

Sets the value of accumulator 1 [A1] to the value on the screen [S] starting in row 1 column 1 for a length of 12 characters.

```
SET F3="Cupertino, CA"
```

Sets the value of field 3 (F3) equal to the character string "Cupertino, CA".



### NONMATCHING LENGTHS

You can write a SET statement in which the right and left sides of the statement are different lengths or don't match the length of the accumulator. For example, the statement

```
SET A1=S(1,1,15)
```

sets accumulator 1 (A1) to the value of the screen area starting in row 1, column 1, for 15 columns [S(1,1,15)]. Accumulator 1 has a length of 12 characters (assuming single-precision accumulators) and the right side of the statement refers to a screen area 15 characters in length. The right and left sides of the SET statement have different lengths, and the right side doesn't match the length of the accumulator.

LIfe-Works handles nonmatching lengths as follows:

- If the left side length is less than or equal to the accumulator length (12 for single-precision accumulators, 24 for double-precision accumulators) the right side value (whether letters or numbers) is placed in accumulator 0 and is treated as a number. Under these conditions, you must take the length of accumulator 0 into account even if you are not setting a value into an accumulator.

- If the right side value is shorter than the length of the accumulator, it is right-justified with leading zeros. For example, the statement

```
1I5 SET S(10,1,10)=F1
```

has a left side length of ten characters [S(10,1,10)] and a right side length of five characters (F1).

If field 1 equals 12345, then after the SET statement executes,

```
accumulator 0 = 000000012345
```

and

```
S(10,1,10) = 0000012345
```

- If the right side value is greater than the accumulator length, the leftmost digits of the right side are cut off. For example, the statement

```
1I15 SET S(10,1,10)=F1
```

has a right side length of 15 (F1) which is greater than the accumulator length. If field 1 equals 123451234512345, then after the SET statement executes,



accumulator 0 = 451234512345

and

S(10,1,10) = 1234512345

- The length of the left side of the SET statement determines how many characters from accumulator 0 are used.

For example, the statement

1I10 SET S(10,1,5)=F1

has a left side length of five characters and a right side length of ten characters. If field 1 equals 1234567890, after the SET statement executes

accumulator 0 = 001234567890

and

S(10,1,5) = 67890

Note that only the five rightmost characters in the accumulator are used to set the value, regardless of how many characters are actually in the accumulator.

- If the left side length is longer than the accumulator length, (12 for single-precision accumulators, 24 for double-precision accumulators) the right side value (whether letters or numbers) is treated as letters. LIFE-Works does not use accumulator 0 for working storage in this situation.
- If the right side length is greater than the left side length, the rightmost characters are cut off. For example, the statement

1I20 SET S(10,1,15)=F1

has a left side length of 15 characters [S(10,1,15)] and a right side length of 20 characters (F1). If field 1 equals 12345678901234567890, then after the SET statement executes

S(10,1,15) = 123456789012345

- If the right side length is less than the left side length, the right side value is left-justified and padded with blanks when it is assigned to the left side of the statement. For example

1I15 SET S(10,1,20)=F1

has a left side length of 20 [S(10,1,20)] and a right side length of 15 [F1]. If field 1 equals 123456789012345, then after the SET statement executes

S(10,1,20) = 123456789012345XXXXXX

- If the right side of a SET statement is a character string, the length of the string must be less than or equal to the left side length. The character string is treated as letters and is padded on the right side with blanks. (This rule does not apply if you are setting an accumulator to equal the value of the right side. See below.) For example, when the statement

SET S(1,1,10)="CATS"

executes, the specified screen position has the value of CATSXXXXXXXX.

- If the right side of a SET statement is a character string and you are setting an accumulator on the left side to equal the value of the right side, the compiler treats the right side value as a number. This means the right side value is first placed in accumulator 0 and is padded on the left with zeros. For example, when the following statement executes

SET A1="DOGS"

accumulator 0 equals 00000000DOGS. This value is then copied to accumulator 1 (A1) as 00000000DOGS.

- If the right side of a SET statement is a character string and is longer than the left side of the statement, the format will not compile.

SUSPEND FORMAT EXECUTION [SLEEP]

The SLEEP command suspends format execution for a specified number of seconds.

The syntax for the SLEEP command is:

SLEEP (expression)

where expression is an absolute value, an arithmetic expression, or a value generated from any of the following sources:

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c and extending for l columns.

Ff refers to field f of the current format.

LFf refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c and extending for l columns.

RS(n,l) refers to a relative screen area in a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to the screen area beginning in row r, column c, and extending for l columns.

X(id,c,l) refers to index set id, beginning in column c and extending for l characters.

.symbol. refers to a name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.

Usage Notes

- SLEEP executes whenever it is encountered.

Examples

SLEEP (10)

Suspends format execution for ten seconds.

SLEEP (A2)

Suspends format execution for the number of seconds specified in accumulator 2.



### DEFINE SUBROUTINE COMMAND [SUBROUTINE]

The SUBROUTINE command defines the beginning of a subroutine and assigns a name to the subroutine.

The syntax for the SUBROUTINE command is:

SUBROUTINE "name"

where name is the name to be assigned to the subroutine. name can be up to nine characters long and must be enclosed in quotation marks.

### Usage Notes

- All subroutines are positioned after the main format code. The general syntax is:

```
$FORMAT 001 $SUB    /* Main format code */
.
.
CALL "ONE"          /* Call first subroutine */
.
.
CALL "TWO"          /* Call second subroutine */
.
.
.
END                /* End of main format */
SUBROUTINE "ONE"    /* Start of first subroutine */
.
.
.
ENDSUB             /* End of first subroutine */
SUBROUTINE "TWO"    /* Start of second subroutine */
.
.
.
ENDSUB             /* End of second subroutine */
```

Note that the \$FORMAT command must include the \$SUB subcommand, and that the END command is used to indicate the end of the main format.

- The following commands are also used with subroutines:
  - ENDSUB marks the end of each subroutine.

- CALL "name" calls the identified subroutine from the main format.

Both of these commands are described in this section.

- The SUBROUTINE command and the ENDSUB command must be in the same job and batch.
- Coding within a subroutine is like coding within a main format, with the following exceptions:
  - Screen areas. Since the cursor position at the start of a subroutine may vary, do not use screen-formatting commands that refer to a specific row and column location. Instead, use the commands that specify a number of columns from the current cursor position. These include Bw, Ew, and f#w. For example, use B5 (blank the screen for five columns from the current cursor position) rather than B2,1 (blank the screen to row 2, column 1).

To identify a screen area within a subroutine, use the RS validation/generation descriptor. RS lets you define a relative screen area--that is, a screen area that starts a specified number of columns from the cursor position when the subroutine was called. RS has the same general function as the S validation/generation descriptor, but RS is used only for screen positions in subroutines.

- Nesting. You can call a subroutine from another subroutine (nesting).
- Recursion. A subroutine can call itself (direct recursion) or can call another subroutine that calls it in turn (indirect recursion). Subroutines can recurse up to 1000 deep. Subroutines that recurse cannot generate any space on the screen.
- Field numbers. You can define up to 50 fields within a subroutine. Fields defined in a subroutine are completely separate from fields defined in the main program or in any other subroutine. You can assign the same field number to a field in the main format and to a field in a subroutine (and to another field in a different subroutine if you wish). Fields defined within a subroutine are called "local" fields.

To refer to a local field within a subroutine, use the LF validation/generation descriptor. LF has the same general function as the F validation/generation descriptor, but LF is used only for fields in subroutines. Local fields can be referred to only within the subroutine in which they are defined; they cannot be referred to from the main format program or from another subroutine.

You can use the F validation/generation descriptor within a subroutine to refer to a field in the main format.

SUPERVISOR MODE COMMANDS [SUPERS AND SUPERM]

The SUPERS and SUPERM commands transfer control from the current format program to a specified job and batch, and execute that job and batch in mode T.

The syntax for the SUPERS and SUPERM commands is:

$\left\{ \begin{array}{l} \text{SUPERS} \\ \text{SUPERM} \end{array} \right\} = \text{reference}$
---

where

SUPERS executes the first record of the specified job and batch, using mode T-S.

SUPERM executes the entire job and batch, using mode T-M.

reference is either the name of the target job and batch, or a location where the target job and batch name can be found, as follows:

"job,batch" identifies the target job and batch. The string must be enclosed in quotation marks.

An refers to accumulator n.

B(c,l) refers to the work buffer, beginning in column c and extending for l columns.

Ff refers to field f of the current format.

LFf refers to local field f of the current subroutine.

Rfn(c,l) refers to the record buffer for MSFP file fn, beginning in column c and extending for l columns.

RS(n,l) refers to a relative screen area within a subroutine, beginning n columns from the initial cursor position and extending for l columns.

S(r,c,l) refers to the screen area beginning in row r, column c, and extending for l columns.

X(id,c,l) refers to index set id, beginning in column c and extending for l columns.

.symbol. refers to a symbolic name previously assigned to a screen area or a portion of an index set record, work buffer, or MSFP record buffer.



Usage Notes

- SUPERS and SUPERM do not require supervisory passwords for their use.
- When transferring control to a job and batch, SUPERS and SUPERM first release the current record, then write it to disk, and finally release the batch. To prevent a disk write, use the NDR command. (See "No Disk Record Command [NDR]" in this section.)
- Unless you are using the NDR command, there must be at least one data character entered in the record before the SUPERS or SUPERM commands execute.
- No data fields should follow either of these commands unless they are a part of an IF or SPLIT statement or in a NDR record.
- If the specified job is password-protected, the operator is asked to enter the password before the transfer takes place.



### VERIFY ON COMMAND [V]

The V command turns the verify feature on. The verify feature makes it possible for an operator to check entered data in key verify mode.

The syntax for the V command is:

V

### Usage Notes

- The V command turns the verify feature on; the V# command turns the verify feature off. Only fields appearing between V and V# can be key verified.
- See also "Verify Off Command [V#]" in this section.

### Examples

P16,ACCOUNT NUMBER: V G10 V#

Prompts the operator to enter an account number. Turns the verify feature on so the account number can be checked in key verify mode, then turns the verify feature off.

IF OUTBAL THEN V ELSE V#

In key verify mode, tests for an out-of-balance condition (OUTBAL). If the batch is out of balance, turns the verify feature on. If the batch is in balance, turns the verify feature off. This coding eliminates unnecessary rekeying by verifying only the fields that are causing the batch to be out of balance.

### VERIFY OFF COMMAND [V#]

The V# command turns the verify feature off. The verify feature makes it possible for an operator to check entered data in key verify mode.

The syntax for the V# command is:

V#

### Usage Notes

- The V command turns the verify feature on; the V# command turns the verify feature off. Only fields appearing between V and V# can be key verified.
- See also "Verify On Command [V]" in this section.

### Examples

P16,ACCOUNT NUMBER: V G10 V#

Prompts the operator to enter an account number. Turns the verify feature on, so the account number can be checked in key verify mode, then turns the verify feature off.

IF OUTBAL THEN V ELSE V#

In key verify mode, tests for an out-of-balance condition (OUTBAL). If the batch is out of balance, turns the verify feature on. If the batch is in balance, turns the verify feature off. This coding eliminates unnecessary rekeying by verifying only the fields that are causing the batch to be out of balance.

Section 7  
Other Commands

SIGHT VERIFY COMMAND [VALID]

The VALID command flags a record for display in sight verify mode.

The syntax for the VALID command is:

VALID

Usage Notes

- The VALID command is not executed in pass 1.

Examples

IF A3!'000000000000' THEN VALID ELSE NULL

If the value of accumulator 3 is not 0, flags the current record for sight verification.

### ZERO ACCUMULATOR COMMAND [Z]

The Z command clears a specified accumulator (fills the accumulator with zeros).

The syntax for the Z command is:

Zn
----

where n is the number of the accumulator to be cleared (0-23).

### Usage Notes

- All accumulators are automatically set to zero at the start of a batch and at the start of verify reconstruct.
- The Z command executes when the record is displayed (pass 1) and when the record is released (pass 2). See Appendix A, "Record Processing" for more information on pass logic.
- Accumulators maintain totals in two ways: record balancing (crossfoot balancing) and batch or total balancing.
  - If you are keeping record balances that are added to or subtracted from another accumulator for batch balancing or batch totals, use the Z command to zero out the record balancing accumulator at the beginning of each new record. Do not clear out totals at the end of a record.
  - Do not use the Z command to clear accumulators containing a batch balance (the highest numbered accumulator in a job) or a batch total. All accumulators are initialized to zero at the beginning of a batch.
- Do not confuse the Z command with the Z modifier. The Z modifier is described in Section 3, "Field Modifiers."









## Section 8 Arithmetic and Logic

### INTRODUCTION

This section describes the use of relational operators, arithmetic operators, logical connectors, conditional statements, branching, conditional indicators, and conditional commands in Life-Works format programming.

### RELATIONAL OPERATORS

Four relational operators are used in Life-Works format programming:

- = (equal to)
- ! (not equal to)
- < (less than)
- > (greater than)

Use relational operators to check the value on the left side of a statement against the value on the right side of the statement (validation). For example, the statement

I4E<6501

means that the 4-character must-fill integer field (I4E) must contain a number less than 6501.

### Usage Notes

- The value on the right side of the statement can be:
  - A numeric constant; for example, I4E>4500
  - A literal character string; for example, G12="Berkeley, CA"
  - A value specified by a validation/generation descriptor; for example, G5=S(2,1,5). See Section 5 for detailed information on validation/generation descriptors.
  - A value resulting from an arithmetic operation. See "Arithmetic Operators" in this section for details.
- When referring to value sets, you can use only the relational operators equal (=) and not equal (!).
- No spaces are permitted immediately before or immediately after a relational operator.



Section 8  
Arithmetic and Logic

- Numeric constants cannot exceed the length of the accumulator (12 if a single-precision accumulator, 24 if a double-precision accumulator).
- Use string constants whenever possible; they use system resources more efficiently than numeric constants.
- See also the discussion of nonmatching lengths in Section 7, "Other Commands."

### ARITHMETIC OPERATORS

Five arithmetic operators are used in LIfE-Works format programming:

- + add
- subtract
- \* multiply
- % general divide
- / divide by a power of 10 (shift right)

#### Usage Notes

- An arithmetic expression can consist of any logical combination of numeric constants and validation/generation descriptors. For example:

I8=(F2+F3)

SET X(333,23,12)=A6-1

I10=B(16,10)\*F3%12

See Section 5 for general information about validation/generation descriptors.

- No spaces are permitted immediately before or after arithmetic operators.
- LIfE-Works uses accumulator 0 for working storage while evaluating an expression. After an expression is evaluated, accumulator 0 will contain the value of the expression.
- The left side of the statement containing an expression must be less than or equal to the length of the accumulator.
- All elements (terms) of an expression must be less than or equal to the length of the accumulator.
- Arithmetic operators can be used in expressions following relational operators.
- You can group arithmetic expressions by using parentheses. Expressions enclosed in parentheses are evaluated first. If you don't use parentheses, the expression is calculated in the order in which it was written. For example, in the expression

2\*(A1+F2)

A1 is first added to F2 and that sum is multiplied by 2.

Section 8  
Arithmetic and Logic

On the other hand, in the expression

$$2 * A1 + F2$$

2 is first multiplied by A1 and that product is added to F2.

- If the product of a multiplication operation is too long for the destination area, LIFE-Works truncates the leftmost digits of the product.
- When doing a general divide (%) the divisor can be a numeric constant, a field, an index set field, an accumulator value, a system keyword, a work area buffer, an MSFP record buffer, or a screen reference.
- Any remainder from a divide operation is discarded.
- The maximum divisor allowed is 100000000.
- Division by zero gives a result of zero.
- The operator / specifies division by a power of 10. It is most commonly used to shorten values generated by system keywords. See Section 5, "Validation and Generation Descriptors," for examples showing the use of the / operator.
- When dividing a negative number by any power of 10, the negative sign is lost.

### LOGICAL CONNECTORS

Use the logical connectors OR (!) and AND (&) to join program elements during validation of data fields forming logical expressions. For example, the statement

I3="000"!="111"

indicates that either 000 or 111 are valid entries. On the other hand, the statement

I3!"000"&!"111"

indicates that neither of these entries are valid.

The symbol & is less commonly used because AND is implied between expressions that don't have any other symbols. For example, the above statement could also be written as

I3!"000"! "111"

### Usage Notes

- Logical OR and logical AND connectors can be used in compound logical expressions. Examples:

I3!"000"&!"111"

1G3 2I4<6501;F1="ABC"

- Logical AND is implied when no connector appears between comparisons in a compound logical expression.
- When LiFE-Works evaluates a logical expression, only enough comparisons are evaluated to establish the result as true or false. However, all arithmetic expressions are evaluated.



### LOGICAL OR (|)

Use the OR symbol (|) to specify multiple value sets or multiple range and value checks.

#### Usage Notes

- In a sequence of comparisons connected by a logical OR connector, no conditions after the first true condition are evaluated. However, any arithmetic expressions in the statement are evaluated.
- In string comparisons, the OR symbol indicates columns not subject to validation. See Section 5, "Validation and Generation Descriptors," for more information.

#### Examples

```
IF F1='1'|='2'|='3' THEN SET X(999,5,1)=6 ELSE NULL
```

If field 1 (F1) is equal to 1 OR equal to 2 OR equal to 3, then set index set number 999, row 5 column 1, to equal 6; otherwise, do nothing. Remember, if field 1 passes one comparison, the rest of the comparisons are not evaluated because the statement already passed the test.

```
G6='|AB|CD'
```

A six-character general field (G6) where columns 1 and 4 are not subject to validation. Only columns 2, 3, 5, and 6 will be validated.

```
A6=V1|=V2
```

Validates accumulator 6 (A6) against two value sets; that is, the value in accumulator 6 must equal a value in value set 1 OR a value in value set 2.

### LOGICAL AND (&)

Use the logical AND symbol (&) when necessary as a delimiter. AND is implied between relationals, so you normally don't have to use it explicitly.

#### Usage Notes

- In a sequence of comparisons connected by a logical AND connector (either actual or implied) no conditions after the first false condition are evaluated. However, any arithmetic expressions in the statement are evaluated.

#### Examples

```
IF F1!'6'&'7'&'8' THEN SET X(999,5,1)=6 ELSE NULL
```

If field 1 (F1) does not equal 6 AND does not equal 7 AND does not equal 8, then set index set number 999, row 5 column 1, to equal 6; otherwise, do nothing. Remember, if field 1 equals 6, then the remaining comparisons are not evaluated because the statement has already failed the test. Note that the AND symbols (&) in this statement are not required; they are included for purposes of illustration.

```
IF F1<'10'>'02'&'80'>'20'&'90'>'85' THEN SET X(999,5,1)=6 ELSE NULL
```

This statement uses both logical OR and logical AND (the AND is implied). If field 1 equals 50, then the first comparison value (10) fails the test and the 02 condition isn't evaluated. The evaluation continues with the second comparison. In this case, field 1 is less than 80 and passes the test; the second value in the comparison (20) is evaluated and also passes the test. Since the entire second condition is true, no further evaluation of the statement takes place.

BRANCHING STATEMENTS

Branching statements perform different actions based on the presence or absence of a stated condition. Life-Works uses the following elements to form branching statements:

- IF...THEN...ELSE statements
- SPLIT statements
- Conditional indicators and commands

### IF...THEN...ELSE STATEMENTS

An IF conditional statement allows different logic paths to be taken based on a test of a condition. If the condition is true, the THEN path is taken. If the condition is false, the ELSE path is taken.

The syntax for an IF statement is:

IF left side  $\left\{ \begin{array}{c} (=) \\ (!) \\ (>) \\ (<) \end{array} \right\}$  right side THEN alternative 1 ELSE alternative 2

### Usage Notes

- An IF statement can test the value of:
  - A location identified by a validation/generation descriptor. For example, IF A3>'250' THEN...
  - A system keyword. For example, IF #RECNO=F4 THEN...
  - A conditional indicator. For example, IF MARKED THEN...
- The left side of an IF statement must be a single item such as A1, F3, or #RECNO.
- The right side of an IF statement can be a single item or an arithmetic expression. (Example: A2+F4/100)
- Note the following rules for using the IF clause:
  - If the field to be tested is the field just before the IF clause, you don't have to specify the field number. Just follow the IF with a relational operator. For example:

I4 IF>'2350' THEN...

- If the IF clause refers to any field other than the one preceding it, you must specify the field number. For example:

2I3 A4 IF F2<'100' THEN G5E ELSE G5S

Remember ASD must be turned on in order for the automatic skip feature in the last statement (G5S) to work.



Section 8  
Arithmetic and Logic

- If the IF statement compares an accumulator to a string constant, the string should match the accumulator length. That is, the string should be 12 characters long for single-precision accumulators and 24 characters long for double-precision accumulators.

If the string isn't as long as the accumulator, the string cannot be compared properly to the figure in the accumulator. If the string is

'7512'

and the accumulator is

000000007512

Life-Works compares the string (7512) to the first four digits of the accumulator (0000). (See the discussion of nonmatching lengths in Section 7, "Other Commands," for more information.) Keep in mind that although a shorter string will compile, the results may not be predictable.

- Note the following about THEN...ELSE alternatives:
  - The THEN...ELSE alternatives must specify the same amount of space on the screen and in the data record. If you use NDR, only the same amount of space on the screen need be specified in each alternative.
  - If there is no action to be taken in a THEN...ELSE alternative, use the NULL command in that alternative.
  - If a path contains more than one statement, enclose it in parentheses. For example
- If you specify a field number in the THEN path, it is illegal to repeat the field number. The field number specified in the THEN path is automatically assigned to the corresponding screen position in the ELSE path. For example the statement

IF A2='000000000000' THEN (B21 ASD G6S) ELSE (B21 A6E)

IF A3='000000000000' THEN 4I5E=V1 ELSE L5GF1\*A4

says that if accumulator 3 equals zeros, then field 4 (4I5) must equal value set 1. Otherwise, field 4 becomes a 5-character left zero-filled field (L5) generated from field 1 and multiplied by accumulator 4 (GF1\*A4). Note that the ELSE path field is not numbered; field 4 is assumed. In any other operations involving this field, either path of the IF statement is regarded as field 4.

- Any field type or command can appear within the THEN...ELSE path, either alone or in combination with other field types and commands.

- Some examples of actions that can be taken by the THEN...ELSE path in a conditional statement are:
  - specify different types of fields or prompts
  - select alternate program levels
  - turn ASD on or off
  - set an accumulator value
- You can use IF statements to display custom error messages on the message line of the screen. These error messages flash and cause the audible alarm to sound (provided your system is configured for the audible alarm feature). Some examples of IF statements using the custom error message feature are:

A1 IF!SA'YN' THEN 'ANSWER MUST BE Y OR N'

I4 IF>'1986' THEN 'YEAR MUST BE 1986 OR EARLIER'

Note the following:

- The operator must press RESET to clear the error message before resuming data entry.
- After the error message is cleared, the cursor is repositioned at the start of the last field entered.
- The text for the custom error message must be enclosed in either single or double quotes and cannot be longer than 80 characters.
- When you include a custom error message in an IF statement, only the THEN path can be used.
- When using the custom error message feature, the field prior to the IF statement must allow data entry.
- When using the custom error message feature, the system "holds" the field prior to the IF statement for conditional testing. If the IF statement refers to any other field, you must specify a field number.
- Do not precede a custom error message with a text field.
- Do not have ASD on if the preceding field has a D, S, or G modifier.
- IF...THEN...ELSE statements can appear on as many lines as necessary. In fact, you can arrange them any way you wish (as long as one or more spaces separate each statement) to make the code easier to read and debug.

### NESTING IF STATEMENTS

IF statements can be nested—that is, either the THEN or ELSE path of an IF statement can contain another IF statement. For example:

```
IF F2='A' THEN (IF F3='B' THEN I5<'54650'
                  ELSE G5F)
                ELSE A5>13
```

Parentheses are required only if the THEN...ELSE path has a compound statement. As shown above, you can use optional parentheses around an IF...THEN...ELSE statement to make the code easier to read.

### Usage Notes

- There must be an equal number of IF...THEN...ELSE clauses in a nested conditional statement. You might find it helpful to make a diagram of your more complex nested conditionals before coding them.
- The operating system editor vi can check to make sure you have an equal number of beginning and ending parentheses. To access vi from LIfE-Works select mode I. See the System V/68 User's Guide for information on vi.

### Examples

```
IF F3='17'      THEN I4G'2394'
                  ELSE (IF F3='22'
                        THEN I4G'7900'
                        ELSE (IF F3='37'
                              THEN G4G'X482'
                              ELSE (ASD G4S)))
```

Note that parentheses are required in the last ELSE path (ASD G4S) because it is a compound statement. The other parentheses are for clarification only.



### SPLIT COMMAND

Like a nested IF statement, the SPLIT command lets a format program take any one of a number of branches. Each SPLIT command consists of n number of statements, each one representing a possible logic path. The branching is determined by the outcome of the arithmetic expression or by a generation descriptor in the SPLIT command.

The format for this command is:

```
SPLIT n WAYS ON value
```

```
    statement 0  
    statement 1  
    statement 2  
    statement 3  
      .  
      .  
      .  
    statement n-1
```

where

n is the number of possible paths of logic (each statement represents one logical path). This number must be a numeric constant from 2 to 60. The first statement is statement zero, so the number of the last statement is the total number of splits, less 1. For example, the statement

```
SPLIT 4 WAYS ON F1
```

would include four statements numbered from 0 to 3.

value is the value represented by a validation/generation descriptor or an arithmetic expression. value determines which logic path the program takes.

### Usage Notes

- The generation descriptors allowed are A, B, F, LF, R, RS, V, and S.
- Each branch following a split command can be either a single statement or a compound statement. (A compound statement has more than one format program element in it.) All compound statements must be enclosed in parentheses.
- Each branch following a split command must define the same amount of space on the screen as well as in the data record. If you use NDR, only the same amount of space on the screen need be defined.



- If two branch statements are identical, use the special USE BRANCH n statement in the second identical statement to avoid duplicating the first branch statement. You cannot enclose the USE BRANCH statement in parentheses. The n referred to in the USE BRANCH statement refers to the number of the statement in the SPLIT command.
- Statement 0 of the SPLIT command is usually the error branch of the SPLIT statement. Statement 0 executes when the value of the arithmetic expression or generation descriptor in the SPLIT command is equal to zero, or is equal to or greater than n. (See the example below.) You should always include an error branch in your code; you can supply a custom message or use the default error message NOT VALID DATA.
- For information on using the SPLIT command with the value set descriptor (V), see Section 9, "Value Sets."

#### Examples

```
P'EXPENSE CODE:' 111E IF='0'|>'4' THEN 'INVALID CODE, REENTER'
SPLIT 5 WAYS ON F1
(P'ERROR CODE      :' L6) /*Split stmt 0--error branch*/
(P' MERCHANDISE:' L6!?NEG<'150000'B1) /*Split stmt 1 */
(P' TRAVEL:' L6!?NEG<'100000'B1) /*Split stmt 2 */
(P' ENTERTAINMENT:' L6!?NEG<'007500'B1) /*Split stmt 3 */
(P' PAYMENT:' L6!?NEGZ1) /*Split stmt 4 */
```

The first line of the example shows the prompt and entry field for EXPENSE CODE. It is a must-enter field and cannot have a value greater than 4. If the operator tries to enter a number larger than 4, the custom error message INVALID CODE, REENTER appears. You must provide some sort of validation in order to get the proper split. The custom error message is helpful to the operator.

The SPLIT command splits five ways on field 1. Each expense code entered causes a specific branch, or split. For example, if the operator enters 3 in field 1, the prompt ENTERTAINMENT: appears on the screen with a 6-character left-zero-filled entry field that cannot have a negative number and must be less than 007500 (L6?NEG<'007500'). The field modifier B1 adds the balance to accumulator 1.

Statement 0 [(P'ERROR CODE :' L6)] is the error branch. Since the first line of the example validates field 1, statement 0 does not execute unless the operator presses the VALID key.

Note that each SPLIT statement is a compound statement and must, therefore, be enclosed in parentheses.

### USING CONDITIONAL INDICATORS AND COMMANDS

Conditional indicators and commands allow the format program to make choices and take alternative actions based on the outcome of logical tests.

Conditional indicators are used in IF statements to test for a specified condition. For example, the following statement tests if the conditional indicator MARKED is true, and if so, performs a screen print:

```
IF MARKED THEN PRINTS ELSE NULL
```

When a conditional indicator is tested in an IF statement, an equal relational (=) is implied. However, the not equal relational (!) can also be used to test a conditional indicator. Thus, IF MARKED and IF!MARKED are both legal constructions.

Conditional commands are used in THEN...ELSE clauses of IF statements. Conditional commands specify an action to be taken based on the results of a test. For example, the following statement marks a document out of balance by using the conditional command MARK if the value in accumulator 2 is not zero.

```
IF A2!'000000000000' THEN MARK ELSE NULL
```

Table 8-1 lists the conditional indicators and commands described in this section.

#### NOTE

Do not use conditional commands associated with verify mode, such as KVD0C, on jobs that are to be trail verified. Since trail verification means that a job is simultaneously accessed in both entry and verify modes, logic paths based on verify mode may not execute properly.

Table 8-1. Conditional Indicators and Commands

Element	Description
BUFFER	A conditional indicator; true if a work buffer has been allocated for the current terminal.
DOCV	A conditional indicator; true if the current terminal is in conditional verify mode.
EXITF	A conditional command; in key verify mode, skips the remaining commands in a format.
KEYVER	A conditional indicator; true if the current terminal is in key verify mode.
KVDOC	A conditional command; puts the current terminal in conditional verify mode.
MARK	A conditional command; marks a document that contains an error.
MARKED	A conditional indicator; true if the document has been marked by the MARK command.
NEWREC	A conditional indicator; true if the current record is a new record and the terminal is in entry mode, find mode, or pass 2 of verify reconstruct.
NULL	A conditional command; specifies that no action is to be taken.
OUTBAL	A conditional indicator; true if the current terminal is in key verify mode and the current batch is out of balance.
PASTWO	A conditional indicator; true if the format is in pass 2.
VRECON	A conditional indicator; true if the current terminal is in verify reconstruct.



WORK BUFFER CONDITIONAL INDICATOR [BUFFER]

Use the BUFFER conditional indicator to determine whether a work buffer has been allocated for the terminal currently running this format program. For example, the statement

IF !BUFFER THEN GETBUF ELSE NULL

checks to see if a work buffer is allocated, and if not, allocates one.

Usage Notes

- Use the GETBUF command to allocate a work buffer. GETBUF is described in Section 7, "Other Commands."



CONDITIONAL VERIFY INDICATOR [DOCVER]

Use the DOCVER conditional indicator to test for conditional verify mode. For example, the statement

IF DOCVER THEN V ELSE V#

turns the verify feature on if the terminal is in conditional verify mode.

Usage Notes

- Use the KVDOC command to start conditional verify mode. See "Conditional Verify Command [KVDOC]" in this section for details.
- DOCVER is typically used to turn the verify feature on immediately before those fields that can cause a document to be out of balance.

EXIT FORMAT CONDITIONAL COMMAND [EXITF]

Use the EXITF conditional command to skip the remaining commands in a format in key verify mode. EXITF can save processing time by eliminating unnecessary verification. For example, the statement

IF MARKED THEN V ELSE EXITF

turns the verify feature on if the document is marked. If the document is not marked, this statement skips the rest of the format.

Usage Notes

- EXITF must be used before any screen display in the format.
- EXITF is ignored in pass 1, in the last record of a batch, or when not in key verify mode.
- See also "Verify On Command [V]" and "Verify Off Command [V#]" in Section 7, "Other Commands."

Section 8  
Arithmetic and Logic

KEY VERIFY CONDITIONAL INDICATOR [KEYVER]

Use the KEYVER conditional indicator to test for key verify mode. For example, the statement

IF !KEYVER THEN Z2 ELSE NULL

zeroes accumulator 2 in entry, find, or verify reconstruct mode, but not in key verify mode.

#### CONDITIONAL VERIFY COMMAND [KVDOC]

Use the KVDOC conditional command to take a terminal out of entry or find mode and put it into conditional verify mode. For example, the statement

```
IF A2="000000000000" THEN NULL ELSE KVDOC
```

tests to see if the value of accumulator 2 is zero. If it is not, the terminal goes into conditional verify mode, and the first record in the document is displayed for verification.

#### Usage Notes

- Conditional verify mode is like key verify mode, but conditional verify mode applies to a whole document instead of an individual record. Conditional verify mode is typically used to correct an out-of-balance condition in a document.
- Fields to be verified should be preceded by the statement:

```
IF DOCVER THEN V ELSE V#
```

This statement turns verify on if the terminal is in conditional verify mode. The DOCVER conditional indicator, which tests for conditional verify mode, is described in this section.

- KVDOC executes only in new records and only in pass 2.
- Use KVDOC with discretion. It puts a heavy load on the system, because it requires that each record between the current record and the previous document header be accessed from the disk.



ERROR CONDITIONAL COMMAND [MARK]

Use the MARK conditional command to mark a document that contains an error. For example, the statement

```
IF A2!'000000000000' THEN MARK ELSE NULL
```

marks the document if an out-of-balance condition exists.

Usage Notes

- The MARK command sets the MARK flag on the current record. If the current record is not a document header record, the system searches backward through the records until it finds a document header record. The system then marks the document header record and all records between the document header record and the current record.
- The MARK command also sets the MARK flag in the batch directory. This allows the marked document to be tested with batch directory scan statements or viewed in the job status display.
- The MARK command is ignored in index set find mode and in pass 1 of an existing record. (See Appendix A, "Record Processing," for more information on pass 1.)
- Use the MARK command with discretion; it puts a heavy load on the system.
- Use the MARKED conditional indicator to test for a MARKED condition. See "Error Conditional Indicator [MARKED]" in this section for details.

ERROR CONDITIONAL INDICATOR [MARKED]

Use the MARKED conditional indicator to test for a MARKED, or error, condition in the document. For example, the statement

IF MARKED THEN V ELSE V#

turns verify mode on if the document is marked.

Usage Notes

- MARKED is true only in entry mode or key verify mode for a document that has been marked by the MARK command. See "Error Conditional Command [MARK]" in this section for details.

Section 8  
Arithmetic and Logic

NEW RECORD CONDITIONAL INDICATOR [NEWREC]

Use the NEWREC conditional indicator to test for a new record in entry or find mode. NEWREC is acknowledged only during entry or find mode or in pass 2 of verify reconstruct. For example, the statement

IF NEWREC THEN ASD ELSE ASD#

allows fields to be automatically duplicated in a new record but not in an existing record.

NULL CONDITIONAL COMMAND [NULL]

The NULL conditional command specifies that no action is to be taken. Use NULL in conditional statements when there is only one logical path to be taken. For example, the statement

IF NEWREC THEN ASD ELSE NULL

turns ASD on if this is a new record; otherwise, takes no action.



OUT-OF-BALANCE CONDITIONAL INDICATOR [OUTBAL]

Use OUTBAL to test for an out-of-balance condition in a batch during key verify mode. For example, the statement

IF OUTBAL THEN V ELSE V#

turns the verify feature on only if the batch is out of balance. This example eliminates unnecessary rekeying by limiting verification to the fields that are causing an out-of-balance condition.

Usage Notes

- OUTBAL is true only when a batch is being key verified and the batch-balance accumulator contains a value other than zero.

PASS 2 CONDITIONAL INDICATOR [PASTWO]

Use the PASTWO conditional indicator to test if the format is processing in pass 2. PASTWO is frequently used in handling index sets or in setting accumulators.

See Appendix A, "Record Processing," for a detailed description of the PASTWO conditional indicator and pass logic.

Section 8  
Arithmetic and Logic

VERIFY RECONSTRUCT CONDITIONAL INDICATOR [VRECON]

Use the VRECON conditional indicator to test for verify reconstruct. VRECON is true only when verify reconstruct is in progress on a batch. For example, the statement

IF VRECON&PASTWO THEN SET A1=0 ELSE NULL

sets accumulator 1 to zero if verify reconstruct is in progress and the format is in pass 2; otherwise, takes no action.







## Section 9 Value Sets

### INTRODUCTION

A value set is a series of single-field items, or values, entered in a standard Life-Works batch and stored on the disk. For example, the items in a value set might look like this:

Berkeley  
Sunnyvale  
Cupertino  
Palo Alto

A format program can use a value set for validation (comparing the value of a field against each item in the value set) or for generation (copying an item from the value set into a field). You can reduce keystrokes and operator errors by validating against or generating from a value set.

A value set is a system element; each value set can be used in any job.

This section explains how to create a value set and how to use value set validation in a branching logic statement. For additional information on value sets, see the following:

- Section 1, "Introduction," explains how to assign a value set to a given job.
- Section 3, "Field Modifiers," describes the Ix modifier, which generates from a value set.
- Section 5, "Validation and Generation Descriptors," describes the V descriptor, which validates against a value set.

### CREATING A VALUE SET

To create a value set:

- 1 Enter the following command into a batch of MASTER or another suitable job. (If the value set contains an item that is longer than 80 characters, use a general-format job that will accommodate the longest item.)

`$VALUES id $SIZE s`

where

id is the 3-digit identifier to be assigned to this value set. id can be a number from 001 to 999.

s is the length of the longest item in the value set. s can be a number from 1 to 1500.

- 2 Press ACCEPT to release the record.
- 3 Enter the items in the value set, one per record, beginning in column 1 of each record. The items can contain letters, numbers, or special symbols. No item can exceed 1500 characters.

To make validation or generation more efficient, enter the most frequently used items first and keep the value set as short as possible. There is no limit to the number of items a value set can have, but lengthy value sets increase search time.

- 4 Release the last record, then return to the record that contains the \$VALUES command.
- 5 Execute the \$VALUES command using mode T-S. (You use mode T-S rather than mode T-M because only one record is actually executed.)

The message REQUEST COMPLETE indicates that the value set has compiled. If you receive an error message, the mistake is probably in the \$VALUES command; correct the command, then reexecute.

To use a value set in a given job, you must assign the value set to a job in the \$DEFJOB statement. See the description of the \$DEFJOB command in Section 1, "Introduction," for details.

### VALUE SET VALIDATION AND SPLIT COMMANDS

You can use the SPLIT command with value set validation to cause a branch based on the results of the validation.

The system uses accumulator 0 to store the results of validation against a value set. Accumulator 0 is set as follows:

- If a match is found, the number of the matching item (not the value of the item) is loaded into accumulator 0. Items in the value set are counted by the system when the search is made, beginning with item number 1.
- If no match is found, or if validation is off when the value set is searched, accumulator 0 is set to zero.

The following example illustrates how the SPLIT command works when based on the value placed in accumulator 0. Assume that the value set is defined to contain five items:

LONDON  
MADRID  
PARIS  
ROME  
STOCKHOLM

The statement

```
G9=V1
SPLIT 6 WAYS ON A0
    statement 0 executed if validation is overridden
    statement 1 executed if LONDON is entered
    statement 2 executed if MADRID is entered
    statement 3 executed if PARIS is entered
    statement 4 executed if ROME is entered
    statement 5 executed if STOCKHOLM is entered
```

specifies that a nine-character general field (G9) must match one of the items in value set 1 (V1). If the operator enters PARIS, the number 3 is placed in accumulator 0 (A0) and statement 3 of the SPLIT command executes. (Note that the "statements" in the above example are for illustration purposes only and are not format code.)

See Section 8, "Arithmetic and Logic," for general information on SPLIT commands; see Section 5, "Validation and Generation Descriptors," for general information on the V descriptor.









## Section 10

### Index Sets

#### INTRODUCTION

An index set is a set of multiple-field records that can be accessed for validation or generation.

Index sets are similar in concept and use to value sets, which are discussed in Section 9. However, index sets are more versatile than value sets. A value set lets you validate against or generate from a list of single items; an index set lets you validate against or generate from any data in any record of a batch. You can also insert, delete, or update index set records under format control.

An index set is made up of records entered into a regular LIfe-Works job and batch, then compiled into a system element. After a batch has been compiled into an index set, its data is available to any number of format programs.

To use an index set, a format first checks the value of a specified field in each record (a key field). When a record containing the desired key-field value is found, the format selects the record. The format can then use any part of the selected record for validation or generation.

The following list represents a typical index set. Each record in the index set contains a part number, an item description, and a unit price:

1098RW	Washer, round	.90
2134CP	Pin, cotter-type	.50
1150BP	Plate, base	1.10
9534FB	Bushing, flat	.45
1366NB	Bolt, nondescript	.65

An index set like this is often used for generation. For example:

1. A format program prompts the operator to enter a part number.
2. The operator enters 1150BP.
3. The format searches the part-number field in each record until it locates the value 1150BP. (In this example, the part-number field is the key field.)
4. When the format program locates a record that contains 1150BP in the part number field, the format selects the record.
5. The format generates the description Plate, base and the price 1.10 from the selected index set record onto the screen and into the current data record.



Section 10  
Index Sets

LIfE-Works index sets can be single-key or multiple-key. A single-key index set has only one key field. A multiple-key index set can have up to 24 key fields.

NOTE

LIfE-Works creates index sets under C-ISAM, a database manager. For information on C-ISAM, see the C-ISAM Indexed Sequential Access Method manual.

### CREATING AN INDEX SET

You can create an index set from any standard LIFE-Works batch that produces a disk record. The batch can contain any number of records, or it can be empty.

Each field to be used as a key field must meet the following criteria:

- The field must appear in each record.
- The field must occupy the same column positions in each record. The columns do not have to be contiguous; for example, you can define a key field that occupies columns 1, 2, and 5.
- The field must be the same length in each record. It must be at least one character long and cannot exceed 180 characters.

### Procedures

To turn a batch into an index set, enter and execute the following supervisory command:

```
$INDSET id $IXBATCH job,batch $KEYFLD [key] start-end [DUPS]  
[$KEYFLD key start-end [DUPS]...]
```

where

id is a three-digit identifier to be assigned to the index set. The identifier can be any number from 000 through 999.

job,batch identifies the batch to be turned into an index set.

key is the name to be assigned to each key field. Key names can be one to six characters long, must start with a letter, and can contain only letters, numbers, and hyphens. key is optional for the first (primary) key field only; it is required for any subsequent key fields.

start-end defines the starting column and the ending column of each key field. See "Usage Notes" below for more information.

DUPS indicates that the key field being defined can contain duplicate values. DUPS is optional; if you do not use it, duplicate key-field values are not allowed. Note that you specify this parameter separately for each key field.

The message REQUEST COMPLETE indicates that the index set has compiled. You can also use mode S-I to verify that the index set has been created.

### Usage Notes

- You can have as many as 1000 index sets on your system (000-999). You can use any number of index sets in a given job, but a given user can have a maximum of 40 index sets active at one time. A user can open the same index set more than once.
- The entire \$INDSET command must fit on one record. If necessary, use the continuation character (\) to continue the command on the next record.
- You can define up to 24 key fields in each index set. The first key field to be defined becomes the primary key field; this key field is the default when no key field is specified in a descriptor or command. You don't have to assign a name to the primary key field, but you must assign names to all other key fields.
- If a key field is one column long, just specify the column number.  
(Example: \$KEYFLD 20)
- A key field can extend over noncontiguous columns and can be made up of as many as eight segments. To define a noncontiguous key field, specify the starting and ending columns for each segment, separating the segments with commas. (Example: \$KEYFLD 10-15,20,22-25,30-32)
- The ISAM flag is set in the batch directory for the specified batch. When the ISAM flag is set, the only other flags that can be set on the batch are DIS, RRF, and RWR. Any attempt to set other flags results in an error.
- All error messages connected with the compilation of an index set are contained in the Life-Works Message Manual. The most common errors are:
  - An index set with the specified identifier already exists.
  - The batch specified in the \$IXBATCH subcommand does not exist, is active, or has already been used to build an index set.
  - A \$KEYFLD subcommand specifies a column that is greater than the last column of the record.
  - There are duplicate values in a no-duplicates key field. If this error occurs, the number of the record containing the duplicate value appears on the message line.
- Once a batch becomes an index set, you cannot update it through entry mode or find mode; any updates must be done through format code or supervisory commands.

You can insert, delete, and revise index set records by using either format commands or supervisory commands. The format commands used with index sets are described in this section; the supervisory commands used with index sets are described in the Life-Works Supervisor's Manual. You must use supervisory commands to create or delete index sets; commands for these operations are also described in the Life-Works Supervisor's Manual.



VALIDATING AGAINST OR GENERATING FROM AN INDEX SET

The index set validation/generation descriptors let you validate or generate from an index set. Table 10-1 lists the index set validation/generation descriptors.

Table 10-1. Validation/Generation Descriptors Used with Index Sets

Descriptor	Description
K	Validates against a specified key field. When a match is found, selects the matching record and gives this format exclusive access to the record.
KS	Validates against a specified key field. When a match is found, selects the matching record and gives this format shared access to the record.
KN	Validates against a specified key field. When a match is found, selects the matching record and gives this format exclusive access to the record. If there is no exact match, selects the record with the next higher key.
KNS	Validates against a specified key field. When a match is found, selects the matching record and gives this format shared access to the record. If there is no exact match, selects the record with the next higher key.
X	Validates against, generates from, or updates a specified index set field. The format must select the index set record by using one of the <u>K</u> descriptors listed above before the <u>X</u> descriptor can be used.

Usage Notes

- The four K descriptors have similar functions. When the text in this section refers to "any K descriptor" or "the K descriptors," any of the four descriptors can be used.
- The K descriptors perform two functions: they access a specified index set record and they perform validation. You must use one of the K descriptors to select a record from an index set before any other index set commands referencing that index set are used. (The one exception is the INSIXR command, which is described later in this section.)



- There are two ways to access an index set: shared access or exclusive access. Shared access means that several operators can simultaneously use the same index set record. Exclusive access means that only one operator can use the selected index set record. Use shared access for validation and generation; use exclusive access when it's necessary to update a record in the index set.
- All elements that refer to a selected index set record (X, BROWSE, NEXT, BACK and so forth) stop the format program unless preceded by one of the K descriptors. Therefore, you must be sure the K descriptor executes when these commands are used.

The K descriptors are validation descriptors; they do not execute when validation is turned off. There are two situations when validation is turned off:

- When the operator presses RETURN without entering the field. For example, in the statement

```
I5=K901 G6GX(901,12,6)
```

if the operator doesn't enter the five-character integer field, validation does not occur, the index set is not selected, and the field starting in column 12 for 6 characters in index set 901 cannot be generated. At this point the format program stops and the following message appears.

```
Index set 901: Referenced but no record selected
```

To make sure the operator enters data in a field, you can make it a must-enter field.

- When the operator displays an existing record during data entry. To be sure that the K descriptor executes, place it in an IF statement, which always executes. (See Section 8, "Arithmetic and Logic," for the rules governing IF statements.) For example, the following statements

```
I5E IF!K901                /*SELECT INDEX SET RECORD*/  
    THEN 'NOT IN INDEX SET'  
    G6GX(901,12,6)          /*GENERATE FIELD*/
```

indicate a 5-character must-enter field that selects a record in index set 901. If the value entered by the operator isn't a key value in the index set, then the custom error message NOT IN INDEX SET appears.

Since the K descriptor is in an IF statement, it always executes and any index set commands following the descriptor [G6GX(901,12,6)] also execute.

You can use the conditional indicator SELREC in an IF statement to test whether an index set record has been selected before trying to access it. SELREC is described in this section.

- All four K descriptors let you do a short key read--that is, they can select a record by validating against an entry that is shorter than the defined key field. For example, the operator can enter the first three characters of a five-character key field. A short key read is useful in combination with the BROWSE command.

The K descriptors cannot validate an entry that is longer than the defined key field.

EXCLUSIVE ACCESS KEY FIELD VALIDATION DESCRIPTOR [K]

Use the K descriptor to select a record with exclusive access and an exact match between the entry field and the key field. Use K when updating records.

The syntax for the K descriptor is:

Kid[(key)]

where

id is the index set identifier.

key is the key field name. key is optional if you are using the primary key field.

Usage Notes

- K can be used on the right side of a comparison (I4=K512) or in an IF statement (I4 IF=K512 THEN...). Only the equal sign (=) or the not equal sign (!) is allowed in the expression.
- When a match is found between the entry field and the key field, the SELREC and EXCLUSIVE conditional indicators are set true, and the SHARED conditional indicator is set false.
- Use the R command to release exclusive access to a record. R downgrades the record to shared access and allows other operators to use the record at the same time. The R command is described later in this section.

SHARED ACCESS KEY FIELD VALIDATION DESCRIPTOR [KS]

Use the KS descriptor to select a record with shared access and an exact match between the entry field and the key field. Use KS for validation where no record updating is done.

The syntax for the KS descriptor is:

KSid[(key)]

where

id is the index set identifier.

key is the key field name. key is optional if you are using the primary key field.

Usage Notes

- The KS descriptor is identical in function to the K descriptor except that shared access is allowed.
- When a match is found between the entry field and the index set, the SELREC and SHARED conditional indicators are set true and EXCLUSIVE is set false.



EXCLUSIVE ACCESS KEY FIELD OR NEXT HIGHEST KEY FIELD VALIDATION DESCRIPTOR [KN]

Use the KN descriptor to select an index set record, or a record with the next highest value, with exclusive access. The KN descriptor in combination with the BROWSE, NEXT, and BACK commands allows the data entry operator to view the records in a specified index set.

The syntax for the KN descriptor is:

KNid[(key)]

where

id is the index set identifier.

key is the key field name. key is optional if you are using the primary key field.

Usage Notes

- LIFE-Works uses the standard ASCII character set collating sequence to determine which record has the next highest value. If the key field value is larger than the last record in the index set, KN selects the last record.
- Use the R command to release exclusive access to a record. R downgrades the record to shared access and allows other operators to use the record at the same time. The R command is described later in this section.

Examples

G6=KN343 BROWSE343

A six-character general field (G6) that selects a record from index set 343 with exclusive access. An exact match is not required. As soon as the index set record is selected, the BROWSE command executes.

SHARED ACCESS KEY FIELD OR NEXT HIGHEST KEY FIELD VALIDATION DESCRIPTOR [KNS]

Use the KNS descriptor to select an index set record, or a record with the next highest value, with shared access. The KNS descriptor in combination with the BROWSE, NEXT, and BACK commands allows the data entry operator to view the records in a specified index set.

The syntax for the KNS descriptor is:

KNSid[(key)]

where

id is the index set identifier.

key is the key field name. key is optional if you are using the primary key field.

Usage Notes

- KNS is identical in function to the KN descriptor except that shared access is allowed.
- LiFE-Works uses the standard ASCII character set collating sequence to determine which record has the next highest value. If the key field value is larger than the last record in the index set, KNS selects the last record.

Examples

G6=KNS343 BROWSE343

A six-character general field (G6) that selects a record from index set 343 with shared access. An exact match is not required. As soon as the index set record is selected, the BROWSE command executes.

#### INDEX SET RECORD FIELD VALIDATION/GENERATION DESCRIPTOR [X]

Use the X descriptor to validate against or generate from a field in an index set record. X must be preceded by a K descriptor that selects the record. Use the X descriptor as follows:

- with a generate modifier (G)
- as a condition in an IF statement
- after a relational in a validation test
- as a term in an arithmetic expression
- in a SET command

The syntax for the X descriptor is:

X(id,c,l)

where

id is the index set identifier.

c is the beginning column of the field.

l is the length of the field in characters.

#### Usage Notes

- The field defined in the X descriptor is not usually the key field.
- Use the X descriptor on the right side of a comparison to validate fields. For example,

G4=X(550,28,4)

is a four-character general field that must match the value in index set 550, starting in column 28 with a length of 4 characters.

- Use the X descriptor to generate fields. (ASD must be turned on for generation to take place.) For example, the statements

```
ASD                      /*TURN ON ASD*/  
I5GX(550,28,5)
```

turn on ASD, then generate (G) a 5-character integer field (I5) from the record currently selected in index set 550, starting in column 28 with a length of 5 characters.



- If the same index set is used more than once in the same format and these records generate or validate non-key fields later in the format, the format should reselect the index set record before each non-key field. Otherwise, if the data entry operator pages back through the records, an entry field could contain data from the wrong record. Consider the following:

```
P'PART NO. 1' 4I7
                IF!K246 THEN 'INVALID PART NO.'
P'QUANTITY  ' 5L6
P'PRICE     ' 6I7GX(246,33,7)
P'PART NO. 2' 7L7
                IF!K246 THEN 'INVALID PART NO.'
```

The operator enters a value for PART NO. 1. The part number is correct; the operator enters the quantity and the price field is generated from index set 246.

The operator then enters a value for PART NO. 2. Another match is made and another record is selected. Suppose that the operator now wants to go back to the QUANTITY field to make a change. As the operator moves back through the record, the PRICE field is changed to reflect the price of the part on the current record. In other words, field 6 now contains the price for the part in field 7.

However, if you reselect the record before generating the PRICE field, the correct price will remain in the field if the operator goes back to the quantity field. Thus, the coding to reselect the record for the price field could be

```
P'PRICE      '      IF F4=K246 THEN 6I7GX(246,33,7)
                                ELSE G7S
```



### USING THE X DESCRIPTOR IN A SET COMMAND

Use the SET command to assign values to fields when updating index set records. For syntax and general rules for using the SET command, see Section 7, "Other Commands."

#### Usage Notes

- Before updating index set records with the SET command, you must obtain exclusive access to that record using the K or the KN descriptor. For example, in the following statements

```
I4=K555
SET X(555,25,12)=A4
```

notice that exclusive access is obtained in the first statement so the SET statement executes properly.

- You can set values in all fields in an index set by using values from other index sets and a single SET statement. For example:

```
I10=KS254          /*OBTAIN SHARED ACCESS TO 254*/
I10=KS637          /*OBTAIN SHARED ACCESS TO 637*/
I10=K423           /*EXCLUSIVELY ACCESS 423*/
```

```
SET X(423,32,10)=X(254,12,2)+X(637,2,7)
RA423
```

In this example, index sets 254 and 637 are accessed by KS descriptors (shared); index set 423 is exclusively accessed by the K descriptor. The SET statement executes properly, updating index set 423 with information obtained from index sets 254 and 637. If index set 423 were not accessed exclusively, the SET statement would not execute. Note that index set 423 is accessed last so exclusive access is as short as possible.

- When a SET statement updates a field in an index set record and that field contains accumulated contents, the SET statement must follow all entry fields keyed by the operator.

Placing the SET statements at the end of a format prevents them from being reexecuted if the operator moves back in the record and then forward again. In the following example, assume that G10 is the last field in the data record.

```
G10      IF EXCLUSIVE438
          THEN SET X(438,8,12)=X(438,8,12)-F6
          ELSE NULL
```

The EXCLUSIVE conditional indicator is described later in this section.

- Group SET statements at the end of the format program. This prevents an operator from having exclusive access to an index set record for a long period of time. This also reduces the possibility of an operator backing up through SET statements.
- Also see the discussion of nonmatching lengths in Section 7, "Other Commands."

COMMANDS USED WITH INDEX SETS

The next several subsections describe the commands used with index sets. Table 10-2 lists and briefly describes these commands.

Table 10-2. Commands Used with Index Sets

Command	Description
BACK	Selects the record before the currently selected index set record.
BROWSE	Lets the operator page through index set records by using the cursor control keys.
DELIXR	Deletes an exclusively accessed index set record.
INSIXR	Inserts a new record into a specified index set. The inserted record is blank except for the key field and is selected with exclusive access.
NEXT	Selects the record following the currently selected index set record.
R	Releases exclusive access to a selected index set record, but retains shared access to the record.
RA	Releases all access to a selected index set record.
SWITCH	Switches from one key field to another within an index set. Applies only to multiple-key index sets. Useful during index-set browsing.



### SELECT THE PREVIOUS INDEX SET RECORD COMMAND [BACK]

The BACK command used in an IF statement starts an automatic search of the selected index set. When the specified record is found, it remains selected until released by the operator. The BACK command functions the same as the BROWSE command except that it does not require operator intervention. BACK is most commonly used for updating index set records.

The syntax for the BACK command is:

BACKid

where id is the index set identifier.

### Usage Notes

- The BACK command must be preceded by a KN or KNS descriptor. The BACK command retains the level of access established by the preceding K descriptor (shared or exclusive).
- The BACK command selects the index set record before the current record in index set id, then goes back to the previous KN or KNS descriptor and executes all the code between the K descriptor and the BACK command, forming a loop. For example, in the following statements

```
P'LOCATION ' 1G12
P'CODE      ' I10E=KNS700
              IF F1!X(700,24,12) THEN BACK700
              ELSE NULL
```

the operator enters a value for the LOCATION in field 1, then selects index set 700 using the key field CODE. The format checks to see if the value entered in field 1 matches the specified field value in index set 700. If the values do not match, the format executes the BACK command. This causes the format program to return to the previous KN or KNS descriptor and reexecute the IF statement using a new record. This loop continues until there is a match and the ELSE path is taken.

If the format program cannot find a match for the LOCATION field, the loop continues until the first record of the index set is reached. This record remains selected and the search stops.

- If the current record is the first record of the index set, that record remains selected and the loop stops. The format program then executes the next statement.
- The BACK command is ignored in existing records.



Section 10  
Index Sets

- It's helpful if you use SET screen commands to display a facsimile of the formatted index set record on the screen for the data entry operator to look at.
- The format code between the BACK command and the KN or KNS descriptor cannot include any commands that move the cursor--prompt (P) commands, entry fields, tab commands (@), blank commands (B), or text fields (T).

### BROWSE THROUGH INDEX SET RECORDS COMMAND [BROWSE]

Use the BROWSE command to let the data entry operator page through an index set, record by record. BROWSE is most commonly used in updating index records.

The syntax for the BROWSE command is:

BROWSEid

where id is the index set identifier.

#### Usage Notes

- The BROWSE command must be preceded by a KN or KNS descriptor. BROWSE retains the level of access established by the preceding K descriptor (shared or exclusive).
- If the operator presses the down arrow key, BROWSE selects the next record. If the operator presses the up arrow key, BROWSE selects the previous record. Each time the operator presses the up arrow or down arrow, the format program goes back to the previous KN descriptor (without backing through any intervening code) to select another record. The format program then executes the code between the KN descriptor and the BROWSE command.

The key field for each record of the index set appears in the message line at the bottom of the screen. (If the key field is longer than 80 characters, the first 80 characters of the key field appear.) When the desired record is found, the operator can:

- Press INDEX to terminate browsing and get shared access to the record (for validation or generation).
- Press HOME to terminate browsing and get exclusive access to the record (for updating).
- Press SHIFT HOME to terminate browsing and release all access to the record.
- It's helpful if you use SET screen commands to display a facsimile of the formatted index set record on the screen for the operator to see.
- The format code between the BROWSE command and the KN or KNS descriptor cannot include any commands that move the cursor--prompt commands (P), entry fields, tab commands (@), blank commands (B), or text fields (T).
- The BROWSE command is ignored in existing records.

Section 10  
Index Sets

Examples

```
P"CODE "  
G5E IF=KNS901  
    THEN BROWSE901  
    ELSE NULL
```

The KNS descriptor selects the index set record with the matching, or next highest, key for the 5-character general field CODE. The record is selected with shared access so several operators can browse through the same record simultaneously.

### DELETE INDEX SET RECORD COMMAND [DELIXR]

Use the DELIXR command to delete exclusively accessed records for specified index sets without operator intervention.

The syntax for the DELIXR command is:

DELIXRid

where id is the index set identifier.

#### Usage Notes

- The DELIXR command must be preceded by the K descriptor in order to have exclusive access to the record.
- Neither the R nor the RA command can appear in the format code between the K descriptor and the DELIXR command.
- The format can delete an index set record even when another user is waiting for access to the record. The waiting user is informed that the record has been deleted.
- The DELIXR command is ignored in existing records.

#### Examples

```
P'ENTER ACCOUNT TO BE DELETED:'      I12
IF!K224 THEN 'ACCOUNT DOES NOT EXIST'
      ELSE NULL
DELIXR224
```

Deletes specified records in index set 224. Assume that a 12-digit account number is the key field for the index set. If the format program matches the account number in the entry field with a key field value in the index set, the entire record containing that account number is deleted.



INSERT INDEX SET RECORD COMMAND [INSIXR]

Use the INSIXR command to insert a record into an index set.

The syntax for the INSIXR command is:

INSIXR(id,l,n)

where

id is the index set identifier.

l is the record length in characters.

n is the program level (1-15).

| key is the name of the primary key field. It cannot be a non-contiguous field.

Usage Notes

- The INSIXR command uses the value of the data field that immediately precedes it as a primary key field value for the inserted record. All other fields in the inserted record are blank. After inserting the new record, INSIXR automatically selects the record with exclusive access. INSIXR is usually followed by SET statements that assign values to the rest of the fields in the new record.
- If the duplicate values option (DUPS) was not specified for the primary key field, you cannot insert a record that has the same primary key value as a record already in the index set.
- If records are to be inserted into a multiple-key index set, all key fields other than the primary key field should allow duplicate values (DUPS).
- The INSIXR command can optionally include the name of the primary key field immediately following the program level number. See "Examples" below for an example.

Examples

I12 IF!K567 THEN INSIXR(567,78,1) ELSE NULL

If no record in the index set contains the specified key field, this statement inserts a new 78-character record into program level 1 of index set 567. The new record is blank except for the key field, which contains the value of the previous field (I12).

G15   INSXR(222,80,1,NAME)

Inserts a new 80-character record into index set 222. The new record is blank except for the key field NAME, which contains the value of the previous field (G15).

### SELECT THE NEXT INDEX SET RECORD COMMAND [NEXT]

Use the NEXT command in an IF statement to search index set records without operator intervention. When the specified record is found, it remains selected until released by the operator. The NEXT command functions the same as the BROWSE command except that NEXT does not require operator intervention. NEXT is most commonly used for updating index set records.

The syntax for the NEXT command is:

NEXTid

where id is the index set identifier.

#### Usage Notes

- The NEXT command must be preceded by a KN or KNS descriptor. NEXT retains the level of access established by the preceding K descriptor (shared or exclusive).
- The NEXT command selects the index set record after the current record in index set id, then goes back to the previous KN or KNS descriptor and executes all the code between the K descriptor and the NEXT command, forming a loop execution. For example, in the following statements

```
P'LOCATION ' 1G12
P'CODE      ' I10E=KNS700
              IF F1!X(700,24,12) THEN NEXT700
              ELSE NULL
```

the operator enters a value for the LOCATION in field 1, then selects index set 700 using the key field CODE. The format checks to see if the value entered matches the specified field value in index set 700. If the values do not match, the format executes the NEXT command. This causes the format program to return to the previous KN or KNS descriptor and reexecute the IF statement using a new record. This loop continues until there is a match.

If the format program cannot find a match for the LOCATION field, the loop continues until there are no more records. The last record in the loop remains selected and the search stops.

- If the current record is the last record of the index set, that record remains selected and the format program executes the next statement.
- The NEXT command is ignored in existing records.
- It's helpful if you use SET screen commands to display a facsimile of the formatted index set record on the screen for the data entry operator to see.

- The format code between the NEXT command and the KN or KNS descriptor cannot include any commands that move the cursor--prompt commands (P), entry fields, tab commands (@), blank commands (B), or text fields (T).



RELEASE EXCLUSIVE ACCESS COMMAND [R]

Use the R command to release exclusive access to a selected index set record. R retains shared access to the record.

The syntax for the R command is

Rid
-----

where id is the index set identifier.

Usage Notes

- After the R command executes, EXCLUSIVE is false; SELREC is true if it was true before R executed; and SHARED is true if SELREC is true.
- All access to a selected record is released at the end of a format, when the operator exits from the current batch, or when the format selects another record in the same index set using another K descriptor.

RELEASE ALL ACCESS COMMAND [RA]

The RA command releases all access (exclusive or shared) to a selected record in an index set. Use the RA command only when a record is not going to be referred to again by the format program.

The syntax for the RA command is:

RAid
------

where id is the index set identifier.

Usage Notes

- After RA executes, EXCLUSIVE, SHARED, and SELREC are all false.
- All access to a selected record is also released at the end of the current format, when the operator exits from the current batch, or when the format selects another record in the same index set with another K descriptor.

### SWITCH KEY FIELD COMMAND [SWITCH]

The SWITCH command lets the format program switch from one key field to another within a multiple-key index set. Use SWITCH in combination with the BROWSE, NEXT, and BACK commands.

The syntax for the SWITCH command is:

```
SWITCH(id,key)
```

where

id is the index set identifier.

key is the name of the key field you are switching to.

### Usage Notes

- SWITCH applies only to multiple-key index sets.
- The SWITCH command must be preceded by one of the K descriptors. SWITCH retains the level of access established by the K descriptor (shared or exclusive).

### Examples

```
P"CITY "  
G20=KNS921(CITY)  
SWITCH(921,NAME)  
BROWSE921
```

Prompts the operator to enter an address and validates the entry against the key field CITY in index set 921. After selecting a record, switches to the key field NAME and lets the operator browse through records in sequence by NAME.

CONDITIONAL INDICATORS USED WITH INDEX SETS

The next several subsections describe the conditional indicators used with index sets. Table 10-3 lists these conditional indicators.

Table 10-3. Conditional Indicators Used with Index Sets

Conditional Indicator	Description
EXCLUSIVE	Indicates whether a record in a specified index set is currently selected with exclusive access.
SELREC	Indicates whether a record in a specified index set is currently selected.
SHARED	Indicates whether a record in a specified index set is currently selected with shared access.

NOTE

See also the descriptions of the VRECON conditional indicator (Section 7, "Arithmetic and Logic") and of the PASTWO conditional indicator (Appendix A, "Record Processing").



EXCLUSIVE ACCESS CONDITIONAL INDICATOR [EXCLUSIVE]

Use the EXCLUSIVE conditional indicator to test whether an index set record is currently selected with exclusive access.

The syntax for the EXCLUSIVE conditional indicator is:

EXCLUSIVEid
-------------

where id is the index set identifier.

Usage Notes

- EXCLUSIVE is true if an index set record has been selected by either the K or the KN descriptor. For example, the following statement

IF EXCLUSIVE845 THEN SET X(845,25,12)=A3 ELSE NULL

confirms that a record from index set 845 has been exclusively selected before setting a field in the record equal to the contents of accumulator 3.

- When EXCLUSIVE is true, SELREC is also true, and SHARED is false.

### INDEX SET SELECT CONDITIONAL INDICATOR [SELREC]

Use the SELREC conditional indicator to test if a record in a specified index set has been selected by any K descriptor. SELREC does not make any distinction between shared or exclusive access.

The syntax for the SELREC conditional indicator is:

SELRECI <sub>d</sub>
----------------------

where id is the index set identifier.

#### Usage Notes

- Whenever there is a possibility that validation override might be attempted on a job using index sets, use SELREC to confirm that an index set record is selected. For example, the statement

```
IF SELREC999 THEN SET S(10,1,12)=X(999,45,12)
```

confirms that a record in index set 999 has been selected (with either exclusive or shared access) before setting the screen to equal a field from the index set record 999.

- As an alternative to testing with SELREC, you can place all references to setting values in index sets in the THEN portion of a conditional index set validation. For example, the statement

```
2I7 IF=K235 THEN G5GX(235,10,5) ELSE G5G"?????"
```

confirms that a record is selected from index set 235 before generating a value from it.

SHARED ACCESS CONDITIONAL INDICATOR [SHARED]

Use the SHARED conditional indicator to test whether an index set record has been selected with shared access.

The syntax for the SHARED conditional indicator is:

SHAREDid
----------

where id is the index set identifier.

Usage Notes

- SHARED is true when a record has been selected with shared access (KS or KNS), or when a record has been selected with exclusive access and then downgraded to shared access (K or KN followed by R).
- When SHARED is true, SELREC is also true, and EXCLUSIVE is false.







## Section 11 Assigning Symbolic References

### INTRODUCTION

Symbols are names that have been assigned to specified column positions in index set records, work area buffers, MSFP record buffers, and screen areas. After you have assigned a symbol to a specific location, you can use the symbol instead of a validation/generation descriptor. Symbols can be easier to remember than validation/generation descriptors, especially if a format refers to a number of different locations.

For example, to generate from an index set field without using symbolic referencing, you use the X validation/generation descriptor with the index set number, the beginning column of the field, and the length of the field. You must repeat all of this information each time you refer to the field. If you use symbolic referencing, you can refer to the field by a meaningful name, such as "BALANCE."

### ASSIGNING SYMBOLS

Symbols are assigned in a separate DECLARATIONS area at the beginning of the format, after the \$FORMAT command but before any other code. The DECLARATIONS area begins with the keyword DECLARATIONS and ends with the keyword END-OF-DECLARATIONS. Within the area, there are up to four sections, which appear in any order. Each section begins with the section keyword. Section keywords are as follows:

<u>Keyword</u>	<u>Purpose of section</u>
INDEX-SECTION	Assigns symbols to index set fields.
BUFFER-SECTION	Assigns symbols to work area buffer fields.
SCREEN-SECTION	Assigns symbols to screen locations.
RECORD-SECTION	Assigns symbols to MSFP record buffer fields.

The connecting hyphen in each keyword is required.

Each section is divided into levels, forming a hierarchy of records, fields, and subfields.

- Level 1 is required. Level 1 declares the length of the buffer, screen, or index set record, as follows:

INDEX-SECTION  
01 INDEX-id G1                      where id is the index set number and 1 is the record length.

BUFFER-SECTION  
01 BUFFER G1                      where 1 is the work buffer size.

SCREEN-SECTION  
01 SCREEN G1                      where 1 is the screen size.

RECORD-SECTION  
01 FILE-fn G1                      where fn is the MSFP logical file number and 1 is the length of the record buffer.

Note that the INDEX and FILE keywords are hyphenated, and that the letter G comes before the length. Leading zeros are optional.

Example:

```
DECLARATIONS
INDEX-SECTION
01 INDEX-897 G40
```

This example declares index set 897, which has 40-character records.

- Level 2 is required. Level 2 assigns symbolic names to specified column positions. Note that although you must include a second level, you don't have to use the number 2 (with or without leading zeroes) to identify it.

For the INDEX-SECTION, BUFFER-SECTION, and RECORD-SECTION, you must list each field in order, starting with column 1. You must account for all column positions within the record or buffer, even if you do not assign a symbol to certain column positions. In other words, the total length of the fields declared in Level 2 must match the length declared in Level 1. Use the keyword FILLER to indicate a field that has no symbol.

Level 2 of the SCREEN-SECTION is handled somewhat differently. See "Assigning Symbols in the SCREEN-SECTION" in this section for details.

Example:

```
DECLARATIONS
INDEX-SECTION
01 INDEX-897                G40
    02 NAME                  G20
    02 FILLER                 G10
    02 CODE2                  G10
```

This index set record has two symbols; NAME and CODE2. NAME is listed first since it is the first field in the record. It occupies columns 1 through 20. The next 10 columns, columns 21 through 30, are not assigned a symbol. The second symbol, CODE2, begins after the FILLER field, and is 10 characters long. It occupies columns 31 through 40. Note that the lengths of the fields in Level 2 add up to 40, the length declared in Level 1.

- The levels below Level 2 are optional. These levels declare subfields (portions of previously declared fields). The total length of the subfields must match the field length.

Example:

```
DECLARATIONS
INDEX-SECTION
01 INDEX-897                G40
    02 NAME                  G20
        03 FIRST             G10
        03 LAST              G10
    02 FILLER                 G10
    02 CODE2                  G10
END-OF-DECLARATIONS
```

In this example, the NAME field is broken into two subfields, FIRST and LAST. Note that the total length of the fields in level 3 adds up to 20, the length of the NAME field.



Section 11  
Assigning Symbolic References

Level numbers must correctly reflect a hierarchical structure. For example:

Incorrect:

```
01 INDEX-123      G80
   02 NAME         G30
       10 FIRST    G10
       10 MID      G5
   03 LAST         G15
02 ADDRESS        G50
```

Correct:

```
01 INDEX-123      G80
   02 NAME         G30
       03 FIRST-MID G15
           10 FIRST G10
           10 MID   G5
       03 LAST     G15
02 ADDRESS        G50
```

The example on the left is incorrect because the total length of all subfields defined under the Level 2 NAME field doesn't equal the length of the NAME field. The example on the right is correct; at each level, the total length of the subfields matches the length of the field.

The rules for assigning symbols are as follows:

- Levels can be numbered 1-99. Level numbers need not be consecutive, but each level should have a higher number than the level it is subordinate to.
- The DECLARATIONS area uses any format or indentation, as convenient. One name or number, however, cannot be split between two records.
- Comments can be included in the DECLARATIONS area, using either the COMMENT command or the /\* and \*/ delimiters.
- Symbols do not have to correspond to the fields in the record; they can refer to one field, part of a field, or several fields. All symbols must refer to contiguous areas.
- A symbol can be up to 12 characters long. Symbols can contain letters, digits, and embedded hyphens, but must begin with a letter.
- Each symbol must be unique within the format. The format aborts if it contains a reference to a symbol that is assigned more than once. (Special provision is made for included batches of symbols, which may contain duplicate names. See "Sharing Symbol Assignments" in this section for details.)
- The following reserved words cannot be used as symbols:

```
BUFFER-SECTION
INDEX-SECTION
SCREEN-SECTION
RECORD-SECTION
```

```
COMMENT
END-OF-DECLARATIONS
INCLUDE
REDEFINES
```

# ASSIGNING SYMBOLS IN THE SCREEN-SECTION

This subsection assumes that you are familiar with the general procedures for assigning symbols. If not, see "Assigning Symbols" in this section.

The SCREEN-SECTION differs from the other DECLARATIONS sections in several ways:

- The SCREEN-SECTION does not require you to account for the area after the last prompt on the screen. The total length of the declared fields can be less than the screen length declared in Level 1.
- The total length of subfields in the SCREEN-SECTION can be shorter than the field length (and will be, if your system uses attribute spaces. See the note below.)
- You can show unused screen space by giving a row and column number after the keyword FILLER, rather than a field length. For example, if the first field to be declared is in row 5, column 1 on the screen, you can show the unused space as follows:

```
02 FILLER          5,1
```

Note that there is no G before the row and column numbers.

In the following example the screen length is 1920 characters (24x80). There are two symbol assignments: TYPE, assigned to a five-character field starting at row 5, column 1, and CLASS, assigned to a five-character field starting at row 5, column 6. (This example assumes a system that doesn't use attribute spaces.)

```
SCREEN-SECTION
01 SCREEN                      G1920
    02 FILLER                  5,1
    02 TYPE                    G5
    02 CLASS                   G5
```

Note that the total length of the fields in Level 2 does not add up to the screen length declared in Level 1. Since no symbols are assigned to the screen after row 5, column 10, the SCREEN-SECTION ends there.

The example above assumes a system that does not use attribute spaces. If your system uses attribute spaces, you must either allow for the attribute spaces or inhibit them. On a system that uses attribute spaces, the example above would look like this:

```
SCREEN-SECTION
01 SCREEN                      G1920
    02 FILLER                  5,1
    02 NAME                    G26      /* 10+1+15=26 */
        03 FIRST              G10
        03 LAST               G15
```

Section 11  
Assigning Symbolic References

The two subfields of the NAME field, FIRST and LAST, are side by side, with an attribute between them. FIRST is 10 characters long, and LAST is 15 characters long. But NAME, in Level 2, is 26 characters long, not 25. NAME has to be 26 characters long to allow for the embedded attribute space.

You can inhibit the attribute space for a given field in the DECLARATIONS area, thereby eliminating the extra space. To inhibit an attribute space, place a minus sign (hyphen) in the field length parameter of the statement that declares the field, as follows:

```
02 NAME      G(-)25
```

The parentheses around the minus sign are required.



### MULTIPLE SYMBOL ASSIGNMENTS

You can assign multiple symbols to a given field. There are two ways to do this. To redefine an entire area, you can define it twice, starting at Level 1. For example:

```

DECLARATIONS
INDEX-SECTION
01 INDEX-897                G40      /*FIRST DEFINITION*/
    02 NAME                  G20
    02 FILLER                 G10
    02 CODE2                  G10
01 INDEX-897                G40      /*SECOND DEFINITION*/
    02 LINE-ONE               G25
    02 FILLER                 G15
  
```

You can also redefine a field by using the REDEFINES keyword. The new definition must be the same level as the previous definition, and it must follow immediately after the previous definition and its subfields. For example:

```

INDEX-SECTION
01 INDEX-123                G80
    02 NAME                  G40      /* FIRST DEFINITION */
        04 FIRST             G20
        04 LAST              G20
    02 NAME-2 REDEFINES NAME /* REDEFINES NAME FIELD */
        03 INITIALS          G5
            05 FIRST-INIT    G1
            05 FILLER        G1
            05 MIDDLE-INIT   G1
            05 FILLER        G2
        03 LAST-2            G20
        03 FILLER            G15
  
```

You can use symbols from both definitions in the format.

Embedded redefinitions (redefinitions inside redefinitions) are allowed, but be careful to keep the level numbers correct.



Section 11  
Assigning Symbolic References

SHARING SYMBOL ASSIGNMENTS

If you have several formats that refer to the index set or MSFP record buffer, or that have the same work buffer contents, you may find it convenient to place the symbol assignments in a separate batch. You can then "share" the symbol assignments by using the INCLUDE command to include the batch in a format.

For example, you might create a batch containing the following statements:

```
INDEX-SECTION
  01 INDEX-897      G40
    02 NAME         G20
    02 ADDRESS      G10
    02 EMPNO        G10
```

To use these symbol assignments in a format, you would enter the following statements in the DECLARATIONS area of the format:

```
DECLARATIONS
INCLUDE "SYMBOL,X897" /*SYMBOL,X897 CONTAINS SYMBOLS FOR XSET 897*/
END-OF-DECLARATIONS
```

Note that the END-OF-DECLARATIONS keyword must be in the main format, not in the included batch.

When several batches of symbols are included in the format, some of the same symbols may be used more than once. If a symbol with more than one assignment is used in the format, the format aborts. However, to make it easier to INCLUDE batches of predefined symbols, the DECLARATIONS area can assign one symbol name to more than one field if the format does not refer to those fields.

Included batches of symbols cannot contain supervisory commands.





Section 12  
Format Programming for Interactive Communications

INTRODUCTION

If your system uses 3270/3274 communications, you can write format programs that send and receive data interactively over communications lines. For example, a typical interactive program might:

1. Prompt the operator to enter a customer's account number.
2. Send the account number to a mainframe computer.
3. Receive the customer's name, address, and current account balance from the mainframe.
4. Prompt the operator to update the account balance.
5. Send the updated record back to the mainframe.

The mainframe cannot write directly to the LIfE-Works database. Instead, the mainframe sends information to the LIfE-Works screen, filling in data fields defined by the format program. The format program can then process the data as desired.

Before you can run an interactive format, you must establish the communications link. To start a session, select a host (mainframe) from the Supervisor Communications Menu. Once you start a session and the format is running, the operator can connect with the mainframe by pressing the ONLINE key, or the format can connect with the mainframe by executing the ONLINE command.

To stop a session, use CTRL U, one of the ESC U sequences, or the Communications Utility Menu. The LIfE-Works Operator's Manual explains the ESC U sequences.



CODING FOR INTERACTIVE COMMUNICATIONS

Table 12-1 describes the format programming elements used in coding for interactive communications.

Table 12-1. Format Programming Elements  
Used in Interactive Communications

Element	Description
ONLINE [hostname]	<p>This command puts the terminal online under format control. You can use this command in place of the operator's ONLINE key. <code>hostname</code> can be any mainframe listed in the Supervisor Communications Menu. If you omit the host name, the ONLINE command connects your terminal with the first mainframe listed on the Supervisor Communications Menu.</p> <p>The ONLINE command connects your terminal with</p> <ul style="list-style-type: none"><li>• the current session. The current session is the session you used most recently.</li><li>• the first active session if there is no current session.</li><li>• a new session. If there are no active sessions with the specified mainframe, ONLINE starts a new session and connects your terminal to it.</li></ul>
ONLINE#	<p>The ONLINE# command leaves the terminal online but prevents any interaction with the mainframe. You can use this command in place of the operator's ONLINE key.</p>
OFFLINE	<p>The OFFLINE command removes the terminal from mainframe control, and ends the session. Use this command to move this terminal from one mainframe to another.</p>

Table 12-1. Format Programming Elements  
Used in Interactive Communications (continued)

Element	Description
SEND code	<p>This command sends attention ID codes to the mainframe. The attention ID codes simulate the function keys used in 3270/3724 communications. <u>code</u> can be any of the following:</p> <p><u>ATTN</u> to send the same code as the ATTN key to the mainframe (for SNA connections only).</p> <p><u>CLEAR</u> to signal the mainframe that the format has cleared the screen.</p> <p><u>ENTER</u> to send the data on the screen to the mainframe.</p> <p><u>SYSREQ</u> to send a system request to the mainframe (for SNA connections only).</p> <p><u>TREQ</u> to send a test request (request for diagnostic test) to the mainframe (for BSC connections only).</p> <p><u>PA1</u> through <u>PA3</u> to perform program access operations defined by the mainframe.</p> <p><u>PF1</u> through <u>PF12</u> to perform program function operations defined by the mainframe.</p> <p><u>PF13</u> through <u>PF24</u> to perform program function operations defined by the mainframe (for SNA connections only).</p>
WAITW	The WAITW command suspends format execution until the mainframe responds.
ONLINE	This conditional indicator is true if the current terminal is online.
D	You must use the D field modifier for each field to be filled in by the mainframe. For general information about the <u>D</u> modifier, see Section 3, "Field Modifiers." Automatic skip-duplicate (ASD) must be on for the <u>D</u> modifier to work.

Section 12  
Format Programming for Interactive Communications

The examples below demonstrate interactive communications with a mainframe.

The following format code sends an entered social security number to the mainframe; the mainframe supplies the name of the person who has that social security number.

```

ASD                      /* TURNS ASD ON                      */
ONLINE                   /* PUTS THE TERMINAL ONLINE */
P"SS #: "                /* PROMPTS OPERATOR TO ENTER SOCIAL SECURITY NUMBER */
G9EF                     /* ENTRY FIELD FOR SOCIAL SECURITY NUMBER */
P"NAME: "                /* SCREEN PROMPT FOR NAME FIELD */
SEND PF1                 /* SENDS SOCIAL SECURITY NUMBER TO MAINFRAME */
WAITW                    /* WAITS FOR MAINFRAME TO SUPPLY NAME */
G20D                     /* ENTRY FIELD FOR MAINFRAME'S RESPONSE */

```

The following format code uses both local and central (mainframe) data files. When the operator enters an account number, the format first checks a local index set. If a matching number is found in the index set, the format uses the index set record to generate additional fields (such as a name and address) into the current data record. If the account number cannot be found in the local index set, the format sends the number to the mainframe. The mainframe then generates the requested fields.

```

ASD                      /* TURNS ASD ON                      */
ONLINE                   /* PUTS THE TERMINAL ONLINE */
P"ACCOUNT #: " I5 B5,1   /* PROMPTS OPERATOR TO ENTER NO. */
IF=K100 THEN             /* CHECKS INDEX SET 100 */
    (G25GX(100,6,25) B6,1 /* IF MATCH FOUND, GENERATES THREE */
     G20GX(100,55,20) B7,1 /*   FIELDS INTO DATA RECORD */
     G25GX(100,76,25))
ELSE                      /* IF MATCH NOT FOUND, SENDS NO. */
    (SEND ENTER WAITW    /*   TO MAINFRAME */
     G25D B6,1           /* MAINFRAME GENERATES THREE FIELDS */
     G20D B7,1           /*   INTO DATA RECORD */
     G25D)               /*   INTO DATA RECORD */

```



### COMMUNICATIONS IN EXISTING RECORDS

The SEND and WAITW commands are normally disabled in existing records, so the operator can page through existing records without communicating with the mainframe. However, there are two important exceptions:

- SEND and WAITW are enabled if an existing record is opened for correction (for example, if the operator presses CORR). This lets the mainframe update the fields that are under its control if a change is made in an existing record.
- SEND and WAITW are enabled if the terminal is in key verify mode and the verify feature is on (the V command has been executed). You can write a format in which information is entered offline and then key-verified online, with the mainframe filling in its fields during key verify. This method helps ensure that data sent to the mainframe is accurate.

The following format code prompts the operator to enter a name and a transaction amount offline. When the record is key verified, the mainframe supplies an account number and the current balance. The format then updates the account balance and sends the updated record back to the mainframe.

```

ASD                                /* TURNS ASD ON                */
ONLINE                            /* PUTS THE TERMINAL ONLINE    */
FDR                               /* FORCES DISK WRITE IN MODE V-K */
E400
P"NAME: " V G2OE @2,1             /* OPERATOR ENTERS NAME OFFLINE */
IF KEYVER THEN                   /* IF IN KEY VERIFY MODE, SENDS  */
    (SEND PF1 WAITW)             /* NAME TO MAINFRAME            */
    ELSE NULL
P"ACCOUNT #:" V# G9D @2,40        /* HOST FILLS IN ACCOUNT NUMBER */
P"BAL: " 1L7D @3,1               /* HOST FILLS IN CURRENT BALANCE */
P"AMT: " V 2L7E                  /* OPERATOR ENTERS AMOUNT OFFLINE */
P"NEW BAL: " IF KEYVER           /* FORMAT CALCULATES NEW BALANCE */
    THEN L7G(F1+F2)              /* SENDS RECORD TO THE MAINFRAME */
    ELSE G7S
IF ONLINE THEN SEND ENTER
    ELSE SEND ENTER

```

### INFORMATION FOR THE MAINFRAME PROGRAMMER

In order for the mainframe computer to share control of the screen with a LiFe-Works format, the mainframe programmer must ensure that:

- The mainframe configuration includes 3277 PU type 2 terminals.
- The mainframe program is in complete agreement with the format program. The mainframe programmer must know exactly how the format is coded. The mainframe must send the appropriate Set Buffer Address (SBA) orders to place fields correctly on the screen.



INFORMATION FOR THE FORMAT PROGRAMMER

Similarly, for successful 3270 interactive communications, as the format programmer you must be sure that:

- The configuration specifies that attributes occupy a screen position.
- The configuration specifies line wrap at column 80.
- The \$FORMAT parameters do not contain the \$ATTRSPACE NO command.

The rest of this section discusses the attribute character and the MDT bit.

Attribute Character

The attribute character (byte) that accompanies each screen element determines display characteristics (as described in Section 4) and also determines how the element is handled by Life-Works and by the mainframe. Table 12-2 explains each bit of an attribute byte. Note that interactive communications supports only 3277 PU Type 2 attributes; it does not support flashing, reverse video, or underscoring.

Table 12-2. Bit Settings in the Attribute Byte

Bit	Value and Meaning
0	Always 1
1	Always 1
2	Protected/unprotected field 0 = Unprotected field 1 = Protected field (P, T, B, or @ commands)
3	Alphanumeric/numeric field 0 = Alphanumeric (field types A, a, G, g, H, N, and w) 1 = Numeric lock (field types I, L, and Lw.d)
4 and 5	Screen intensity 00 = Normal (low) 01 = Normal (low) 10 = High 11 = Blank (non-display and non-print)
6	Text/non-text field 0 = Not a T field 1 = T field
7	Modified data tag (MDT) 0 = Cleared (field not entered or changed) 1 = Set (field entered or changed)

#### Displaying the Attribute Character

Displaying the attribute characters shows you which bits are set in the attribute byte. To do this, press HELP. A character appears before each prompt and entry field. Each character corresponds to one combination of bit settings.

Table 12-3 lists each character and the corresponding bit settings. In the table, the column headings refer to the following bits:

<u>Column Heading</u>	<u>Bits</u>
U/P	2
A/N	3
Inten	4 and 5
Text	6
MDT	7

Section 12  
Format Programming for Interactive Communications

Table 12-3. Display of Attribute Bytes

HELP Display	U/P	A/N	Inten	Text	MDT	HELP Display	U/P	A/N	Inten	Text	MDT
@	0	0	00	0	0	`	1	0	00	0	0
A	0	0	00	0	1	a	1	0	00	0	1
B	0	0	00	1	0	b	1	0	00	1	0
C	0	0	00	1	1	c	1	0	00	1	1
D	0	0	01	0	0	d	1	0	01	0	0
E	0	0	01	0	1	e	1	0	01	0	1
F	0	0	01	1	0	f	1	0	01	1	0
G	0	0	01	1	1	g	1	0	01	1	1
H	0	0	10	0	0	h	1	0	10	0	0
I	0	0	10	0	1	i	1	0	10	0	1
J	0	0	10	1	0	j	1	0	10	1	0
K	0	0	10	1	1	k	1	0	10	1	1
L	0	0	11	0	0	l	1	0	11	0	0
M	0	0	11	0	1	m	1	0	11	0	1
N	0	0	11	1	0	n	1	0	11	1	0
O	0	0	11	1	1	o	1	0	11	1	1
P	0	1	00	0	0	p	1	1	00	0	0
Q	0	1	00	0	1	q	1	1	00	0	1
R	0	1	00	1	0	r	1	1	00	1	0
S	0	1	00	1	1	s	1	1	00	1	1
T	0	1	01	0	0	t	1	1	01	0	0
U	0	1	01	0	1	u	1	1	01	0	1
V	0	1	01	1	0	v	1	1	01	1	0
W	0	1	01	1	1	w	1	1	01	1	1
X	0	1	10	0	0	x	1	1	10	0	0
Y	0	1	10	0	1	y	1	1	10	0	1
Z	0	1	10	1	0	z	1	1	10	1	0
/	0	1	10	1	1	{	1	1	10	1	1
*	0	1	11	0	0		1	1	11	0	0
!	0	1	11	0	1	}	1	1	11	0	1
^	0	1	11	1	0	~	1	1	11	1	0
_	0	1	11	1	1	&	1	1	11	1	1

Press any key to remove the attribute characters.

#### MDT Bit

Bit 7 of the attribute byte has particular significance for interactive programming. This bit, called the modified data tag (MDT) bit, determines which of the elements on the LiFE-Works screen can be sent to the mainframe. If the element's MDT bit is set (has a value of 1), the element is considered to be data and can be sent to the mainframe. If an element's MDT bit is cleared (has a value of 0), the element is considered to be a prompt and cannot be sent to the mainframe.



LIfe-Works handles the MDT bit as follows:

- In new records, the MDT bit is initially cleared for both prompts and data fields. The MDT bit is set when:
  - The operator keys data into a field. (The RIGHT ARROW and LEFT ARROW keys do not alter the MDT bit.)
  - Data is placed in a field through generation, duplication, or format code commands (for example, SET F1=A2).
  - Format code commands are executed to set the bit.
- In existing records, the MDT bit is set for all entry fields.

The following format commands affect the MDT bit:

- SETMDT Ff sets the MDT bit for field f.
- CLEARMDT Ff clears the MDT bit for field f.
- CLEARMDT ALL clears the MDT bit for all elements on the screen.
- EUAr,c and EUAw erase all unprotected fields and clear their MDT bits, starting from the cursor position and extending to the position specified by r,c or w. (Unprotected fields are those in which bit 2 of the attribute byte is cleared—usually data entry fields.) The EUA commands are described in Section 6, "Screen-Formatting Commands."
- RESETMDT erases text fields and clears their MDT bits.

The operator's ERASE INPUT key erases unprotected fields and clears their MDT bits, from the cursor position to the end of the screen. Note that the LIfe-Works ERASE INPUT key differs slightly from the ERASE INPUT key on a 3270 system, which erases all unprotected fields on the screen.

The mainframe can clear all MDT bits by using the Reset MDT function in the write control character, or can clear the MDT bit from unprotected fields by issuing an Erase All Unprotected command.

Fields placed on the screen by a T (text) command are protected and have the MDT bit set. Text fields are not erased by a mainframe Erase All Unprotected command, an EUA format command, or the ERASE INPUT key. However, text fields are erased and their MDT bits cleared by the mainframe Reset MDT command or the RESETMDT format command.

LIfe-Works does not check or change the MDT bit when it writes a record to disk.



ERROR CONDITIONS

LiFE-Works notes any error condition in the LiFE-Works log. When an error occurs, a message appears at the bottom of the screen referring you to the LiFE-Works log. Have your system administrator look up the error condition code and message in the log file (\$W/text/log) and report the error to your service representative if you can't resolve the problem.





### Section 13 The Data Dictionary

The LiFE-Works data dictionary allows other software products, such as LiFE-Forms and LiFE-Plans, to use data from the LiFE-Works database.

The data dictionary is made up of data definitions. You must create a data definition to describe each LiFE-Works job, index set, or MSFP Ftype that contains data to be used in other software products. A data definition contains the following information:

- The type of data being defined (ordinary job, index set, or MSFP Ftype).
- The location of the data (the job name, index set identifier, or Ftype identifier; the program level; and the node name).
- A series of field definitions. Each field definition assigns a name to a field in the data record and provides information about that field.

To create a data definition, you fill out a set of menus. Table 13-1 describes the keys used in the data dictionary menus.

Table 13-1. Keys Used in Data Dictionary Menus

Key	Function
RIGHT ARROW LEFT ARROW UP ARROW DOWN ARROW	Move the cursor from item to item within a menu.  <u>NOTE:</u> Some menus have a fill-in area and a selection area. When the cursor is in the fill-in area, pressing DOWN ARROW moves the cursor into the selection area. You can then use the arrow keys to move through the selections. To move the cursor back into the fill-in area, press HOME.
TAB	Moves the cursor to the beginning of the next field.
SHIFT TAB	Moves the cursor to the beginning of the previous field.
RETURN	Blanks the rest of the field and moves the cursor to the next field.
DOC FWD	In menus that contain a list of names, displays the next screenful of names.



Table 13-1. Keys Used in Data Dictionary Menus (Continued)

Key	Function
DOC BACK	In menus that contain a list of names, displays the previous screenful of names.
HOME	In menus that have a fill-in area and a selection area, moves the cursor back from the selection area into the fill-in area.
SELECT	In menus that allow multiple selections, selects the item at the cursor position.
CTRL SELECT	Cancels the selection of a menu option that you chose with SELECT.
ACCEPT	Indicates that a filled-in menu is complete. In menus that allow only one selection, ACCEPT accepts the item at the cursor position and displays the next menu.
SHIFT DELETE	Deletes a specified data definition or field definition.
EXIT	Moves back to the previous menu. If the DATA DICTIONARY GENERATOR menu is on the screen, EXIT stops the data dictionary generator.  In general, EXIT cancels any changes you have made in the current menu. Two exceptions are the ADD NEW FIELD DEFINITIONS menu and the CHANGE A FIELD DEFINITION menu. Field definitions are stored temporarily as you create them; when you exit from the field definition menus, any new or changed field definitions remain in temporary storage.
CTRL EXIT	Cancels any changes in the current menu, and moves back to the DATA DICTIONARY GENERATOR menu. If the DATA DICTIONARY GENERATOR menu is on the screen, CTRL EXIT stops the data dictionary generator.

### STARTING THE DATA DICTIONARY GENERATOR

The data dictionary generator lets you create, modify, and delete data definitions.

To start the data dictionary generator from inside LIFE-Works:

- 1 If necessary, press CTRL Z to display the SUPERVISOR MENU.
- 2 Select Application Development from the SUPERVISOR MENU.
- 3 Select Generate Data Dictionary from the APPLICATION DEVELOPMENT menu.

To start the data dictionary generator from the operating system, execute the command:

LWddg

The DATA DICTIONARY GENERATOR menu appears (Figure 13-1). This menu is the starting point for creating a new data definition, viewing or modifying an existing data definition, or deleting a data definition. Each of these operations is described in this section.

DATA DICTIONARY GENERATOR			
Data Definition Name: _____			
<input type="radio"/> name	<input type="radio"/> name	<input type="radio"/> name	<input type="radio"/> name
<input type="radio"/> name	<input type="radio"/> name	<input type="radio"/> name	<input type="radio"/> name
.	.	.	.
.	.	.	.
.	.	.	.

Figure 13-1. The DATA DICTIONARY GENERATOR Menu

### CREATING A NEW DATA DEFINITION

To create a new data definition:

- 1 Start the data dictionary generator as described earlier in this section.
- 2 When the DATA DICTIONARY GENERATOR menu appears, type a name for the new data definition in the Data Definition Name: area. The rules for data definition names are:
  - A name can be up to 14 characters long.
  - A name can contain any combination of letters, numbers, and underscores (SHIFT HYPHEN), but it must begin with a letter.
  - Each name must be unique within the data dictionary.
  - You can use both upper- and lowercase letters. Case is significant; vendor, Vendor, and VENDOR are three different data definition names.

#### NOTE

Future releases of LIFE-Works may offer a data dictionary interface to INFORMIX. If you plan to use this feature, be aware that INFORMIX does not distinguish between upper- and lowercase letters; total, Total, and TOTAL are the same name to INFORMIX.

- 3 Press ACCEPT. The DATA DEFINITION menu appears (Figure 13-2).

DATA DEFINITION

Definition name: name

Select the data type:

- o Job
- o Index Set (1)
- o MSFP File Type
- o Define or view field definitions (2)

Program level: 01 (3)

Figure 13-2. The DATA DEFINITION Menu



4 Fill in the DATA DEFINITION menu as follows:

- ① Select the data type: Move the cursor to the desired selection, then press SELECT.

After you select the data type, LIFE-Works prompts you for one of the following:

- A job name and optional OfficeLAN node name.
- An index set identifier and optional OfficeLAN node name.
- An MSFP Ftype identifier.

Enter the requested information, then press ACCEPT to return to the DATA DEFINITION menu. If you don't enter a node name for a job or index set, LIFE-Works assumes that the data is on your own system.

Note that you can create a data definition for a job, batch, or index set that does not exist yet.

- ② Define or view field definitions: This item lets you create new field definitions or view existing definitions. To select this item, move the cursor to the item, then press SELECT. Field definitions are described below.
- ③ Program level: Enter the program level number of the data being defined. The default program level is 01. To indicate any other program level, enter a number (02-15) over the default number. To indicate that the definition applies to all program levels, enter 00.

5 When you select "Define or view field definitions" for a new data definition, the ADD NEW FIELD DEFINITIONS menu appears (Figure 13-3).

6 Fill in the ADD NEW FIELD DEFINITIONS menu as follows:

- ① Field name: Type a name for the field being defined. The rules for field names are:
  - A name can be up to 14 characters long.
  - A name can contain any combination of letters, numbers, and underscores (SHIFT HYPHEN), but it must begin with a letter.
  - Each name must be unique within the data definition.
  - You can use both upper- and lowercase letters. Case is not significant; total, Total, and TOTAL are the same field name.



ADD NEW FIELD DEFINITIONS

Field name: \_\_\_\_\_ ①

Starting column: ② \_\_\_\_\_

Field length: ③ \_\_\_\_\_

Decimal places: ④ \_\_\_\_\_

Select the field type: ⑤

o General      o Date

o Integer      o Currency

o Field is an index set key field ⑥

Figure 13-3. The ADD NEW FIELD DEFINITIONS Menu

- ② Starting column: Enter the starting column of the field being defined.
- LIFe-Works supplies a starting column number, beginning with column 1 and incrementing to the next undefined column number as you create each field definition. If the supplied column number is correct, move on to the next field; if not, you can type over the number.
- ③ Field length: Enter the length of the field in columns.
- Date fields (described below) are always eight characters long. If you choose this field type, you can leave the "Field length" item blank; LIFe-Works automatically supplies a value of 8.
- ④ Decimal places: Enter the number of digits to the right of the decimal place. This item applies only to integer and currency fields.
- ⑤ Select the field type: Move the cursor to the desired selection, then press SELECT. The field types and the characters they can contain are:
- General: Any characters.
  - Integer: Numbers 0-9 and optional leading or trailing blanks.
  - Date: A date in the format mm/dd/yy (month/day/year).
  - Currency: Numbers 0-9 and optional leading or trailing blanks.

- ⑥ Field is an index set key field: Select this item to indicate that the field being described is a key field. This selection applies only to index sets (including MSFP XD files).
- 7 When you finish the field definition, press ACCEPT. A blank ADD NEW FIELD DEFINITIONS menu appears, and the completed field definition appears beside it.
- 8 Repeat steps 5 through 7 above until all desired fields are defined.
- 9 Press EXIT. The FIELDS DEFINED FOR DATA DEFINITION name menu appears (Figure 13-4), listing all field definitions for this data definition.

FIELDS DEFINED FOR DATA DEFINITION <u>name</u>							
Field name: _____							
Field Name	Col	Type	Key	Field Name	Col	Type	Key
o				o			
o				o			
o				o			
o				o			
o				o			

Figure 13-4. The FIELDS DEFINED FOR DATA DEFINITION name Menu

- 10 Press EXIT again to reach the DATA DEFINITION menu.
- 11 Press ACCEPT to write the data definition to disk. Note that field definitions are not stored permanently until you complete this step.

You can also create field definitions by using a free-form entry method. To use free-form entry:

- 1 Display either the FIELDS DEFINED FOR DATA DEFINITION name menu or the ADD NEW FIELD DEFINITIONS menu.
- 2 Press CTRL Z. The following prompt appears:

field name,column,type,key: \_\_\_\_\_

- 3 Enter the field definition, using the following syntax:

name,column,type,[,K]

where

name is the field name. Rules for field names appear earlier in this section.

column is the starting column of the field in the record.

type is the field type and length. type can be one of the following:

Gw to indicate a general field (any characters). w is the length of the field in characters.

Iw to indicate an integer field (integers and optional leading or trailing blanks). w is the length of the field in characters.

Iw.d to indicate an integer field (integers and optional leading or trailing blanks) with an implied decimal point. w is the length of the field in characters; d is the number of characters to the right of the decimal point.

Cw to indicate a currency field (integers and optional leading or trailing blanks). w is the length of the field in characters.

Cw.d to indicate a currency field (integers and optional leading or trailing blanks) with an implied decimal point. w is the length of the field in characters; d is the number of characters to the right of the decimal point.

D8 to indicate a date field in the format mm/dd/yy (month/day/year). The eight-character length is required.

K indicates that this field is an index set key field.

Examples of free-form field definitions:

NAME,1,G20,K

ADDRESS,30,G20

ZIP,60,I5,K

SALARY,65,I10.2

- 4 Press ACCEPT to complete the free-form field definition. A blank free-form menu appears, and the completed field definition appears below the menu.



- 5 Repeat steps 1 through 4 above until all desired fields are defined.
- 6 Press EXIT twice to return to the DATA DEFINITION menu.
- 7 Press ACCEPT to write the data definition to disk. Note that field definitions are not stored permanently until you complete this step.



### MODIFYING A DATA DEFINITION

To modify an existing data definition:

- 1 Start the data dictionary generator as described earlier in this section.
- 2 When the DATA DICTIONARY GENERATOR menu appears, do one of the following:
  - Type the name of the data definition to be modified in the entry area, then press ACCEPT.

or

- Move the cursor to the name of the data definition to be modified, then press ACCEPT.

The DATA DEFINITION menu appears, displaying the current information for the data definition you specified.

- 3 To change any of the displayed information, type a new entry over the existing entry.

- 4 To modify any of the field definitions for this data definition:

4A Move the cursor to Define or view field definitions on the DATA DEFINITION menu, then press SELECT. The FIELDS DEFINED FOR DATA DEFINITION name menu appears, listing all defined fields.

4B Do one of the following:

- Type the field name after the Field name: prompt, then press ACCEPT.

or

- Move the cursor to the field name in the FIELDS DEFINED FOR DATA DEFINITION name menu, then press ACCEPT.

4C The CHANGE A FIELD DEFINITION menu appears, displaying the current definition for the field you selected. To change any of the displayed information, type a new entry over the existing entry. (Note that you can't type over the field name; to change a field name, you must delete the field definition, then create a new one using the desired name.)

4D Press ACCEPT to store the completed field definition temporarily. The FIELDS DEFINED FOR DATA DEFINITION name reappears.

4E Repeat steps 4B through 4D as many times as necessary.

4F Press EXIT to return to the DATA DEFINITION menu.

- 5 Press ACCEPT to write the modified data definition to disk.

### DELETING A DATA DEFINITION

You can delete a data definition from either the DATA DICTIONARY GENERATOR menu or the DATA DEFINITION menu.

To delete a data definition while the DATA DICTIONARY GENERATOR menu is on the screen, do one of the following:

- Move the cursor to the name of the data definition to be deleted, then press SHIFT DELETE.

or

- Type the name of the data definition in the entry area, then press SHIFT DELETE.

To delete a data definition while the corresponding DATA DEFINITION menu is on the screen, press SHIFT DELETE. (The cursor can be positioned anywhere in the menu.)

In all cases, LIFE-Works asks you to verify that you want to delete the specified data definition.

#### DELETING A FIELD DEFINITION

You can delete a field definition from either the FIELDS DEFINED FOR DATA DEFINITION name menu or the CHANGE A FIELD DEFINITION menu.

To delete a field definition while the FIELDS DEFINED FOR DATA DEFINITION name menu is on the screen, do one of the following:

- Move the cursor to the name of the field definition to be deleted, then press SHIFT DELETE.
- or
- Type the name of the field definition to be deleted in the entry area, then press SHIFT DELETE.

To delete a field definition while the corresponding CHANGE A FIELD DEFINITION menu is on the screen, press SHIFT DELETE. (The cursor can be positioned anywhere in the menu.)

In all cases, LiFE-Works asks you to verify that you want to delete the specified data definition.

### COPYING A DATA DEFINITION

You can copy a data definition to another data definition or to an operating system file. The syntax is:

$\begin{array}{l} \$COPY \left\{ \begin{array}{l} \$DD \text{ [node] name} \\ \text{pathname} \end{array} \right\} \left\{ \begin{array}{l} TO \\ OVER \end{array} \right\} \left\{ \begin{array}{l} \$DD \text{ [node] name} \\ \text{pathname} \end{array} \right\} \end{array}$
--

where

node is an OfficeLAN node name. If you don't specify node, your own system is assumed.

name is the source or the target data definition name.

pathname is the full pathname of the source or target operating system file.



1000

1000

1000





Section 14  
VISION II Format Generator

INTRODUCTION

The LiFE-Works format generator lets you develop LiFE-Works formats without writing format code. The format generator produces a completed format program that you can compile, use, and edit as if you had written it in format language. The information in this section assumes that you have a basic understanding of format programming concepts.

When creating a new format, you enter the prompts and the fields directly onto your screen, where they have the same position and length as they do on the data entry screen. You can use the format generator's function keys to reposition and edit the prompts and the fields.

Once you have created a data field, you can use a menu system to define the field. The first menu contains the following options for field definition: Type, Attribute, Characteristics, Validation, and Processing. Each option brings up another menu containing the possible options for that definition.

The format generator converts the screen image and the field definitions into format code and saves the code in the specified batch. Once this is done, you can compile the format program.

You can also use the format generator to edit simple format programs. When you edit an existing format program, the format generator displays a screen image based on the source code. You enter and delete the prompts and the fields in the same way you would if you were creating a new format.

If you are an experienced format programmer, the format generator provides an easy way to create a foundation for more complex format code. You do not have to compile the code to see what the data entry screen will look like, so you can use the format generator to visually lay out the screen until it is just like you want it. Once the format generator writes the source code into a batch, you can add format code that performs MSFP (Multistation File Processing) and other advanced functions.



### STARTING THE FORMAT GENERATOR

To start the format generator from the menu system:

- 1 Select the Format Generator option from the Applications Development menu.
- 2 Your screen now contains the Define a Format menu shown in Figure 14-1.
- 3 Enter the three-digit identifier of the format you want to generate or edit.
- 4 Enter the job and batch in which you want to store the source code.
- 5 Press ACCEPT to start the format generator.

To start the format generator from the status line:

- 1 Select mode G. LIFE-Works prompts you for the job,batch in which you want to store the source code.
- 2 Enter the desired job,batch. The Define a Format menu (Figure 14-1) appears with the job,batch you specified in the Place Source In Job,Batch field.
- 3 Enter the three-digit identifier of the format you want to generate or edit.
- 4 Press ACCEPT to start the format generator.

Define a Format

Format \_\_\_\_

Place Source In Job,Batch:

\_\_\_\_\_

Figure 14-1. Format Generator Menu

If you enter a job,batch that contains something other than the source code for the format you specified, no prompts or fields appear on the screen. The format generator appends any code you create to the end of the batch.

When you enter a job, batch that contains source code for the specified format, the format generator tries to generate a screen image from the code. If the format generator is able to generate a screen image, you can edit the prompts and fields just like you would if you were creating a new format.

The format generator may not understand the source code. This can happen if the format generator did not create the source code, or if someone edited the code after the format generator created it. If this is the case, the message

Record n: unsupported statement

flashes at the bottom of your screen. n is the number of the first record the format generator cannot read. Press RESET to continue. You cannot edit this batch of source code with the format generator.

### DESIGNING A DATA INPUT SCREEN

The screen you see after the Format Generator menu has a status line at the bottom. Figure 14-2 shows an example status line.

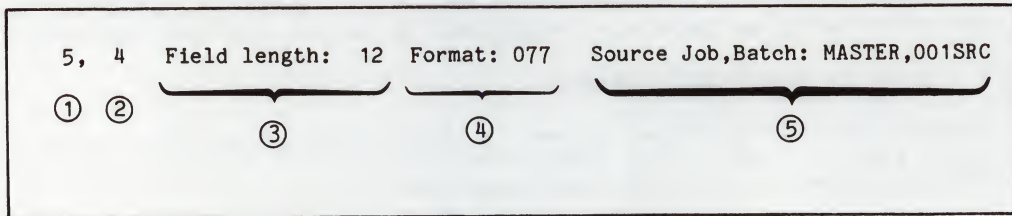


Figure 14-2. Status Line

#### Explanation:

- ① Cursor's row position
- ② Cursor's column position
- ③ Length of the field the cursor is on. Field length is zero if the cursor is not on a field.
- ④ Identifier of the format being generated
- ⑤ Job,batch containing the source code

#### NOTE

You can remove the status line if it is in the path of your screen layout or if you want to increase the speed at which the cursor moves. The status line slows the cursor because the format generator updates the status line each time you move the cursor. Use the LINES key to turn the status line on or off.

Table 14-1 lists the function keys you can use with the format generator. The functions are similar to those used in mode M.



Table 14-1. Format Generator Function Keys

Key	Function
ACCEPT	Tells the format generator that you've completed a menu.
CASE	Toggles between uppercase mode and lowercase mode. Uppercase mode causes all the letters you type to appear in uppercase. Lowercase mode lets you type both upper- and lowercase letters.
CORR	Saves the new format code without exiting the format generator.
DELETE	Deletes the character the cursor is on and moves the characters to the right of the cursor one position to the left.
DESELECT	Deselects a menu item.
DUP	Inserts a line below the current line and copies the current line into the new line.
ERASE	Erases everything from the cursor to the end of the screen.
ERASE EOF	Erases everything from the cursor to the end of the line.
EXIT	Saves the new format code and exits the format generator.
FIRST REC	Moves the cursor to row 1, column 1.
HOME	Moves the cursor to the beginning of the current line.
INSERT	Inserts one space to the left of the cursor.
LINES	Removes or redisplay the sataus line.
RETURN	Moves the cursor to the beginning of the next line.
RECORD DELETE	Deletes the line the cursor is on and moves the lines below the cursor up one position.
RECORD INSERT	Inserts one blank line below the current line.
SELCT	Selects a menu item when you are in a selection menu.  Marks the beginning or the end of the field when you are creating a data field.  Displays the Field Definition menu when the cursor is in an existing data field.



Table 14-1. Format Generator Function Keys (Continued)

Key	Function
SHIFT TAB	Moves the cursor to the beginning of the previous field.
TAB	Moves the cursor to the beginning of the next field.

#### Creating Screen Prompts

Move the cursor to the place you want the prompt and type the prompt. You can modify screen prompts using the function keys explained in Table 14-1.

The format generator uses the P (prompt display) command to convert all the text you type on the screen into prompts. See Section 6, "Screen-Formatting Commands," for details on the P command.

#### Creating Data Fields

Use the following procedure to create data fields:

- 1 Move the cursor to the place you want to begin the field.
- 2 Press SELECT to highlight the cursor's current position.
- 3 Press the RIGHT ARROW to enlarge the field one character position at a time until it is the length you want. The highlighted area marks the exact size and location of the field. Press LEFT ARROW to reduce the field size if you accidentally make it too large.
- 4 Press SELECT at the place you want to end the field.

After you have created a data field, you can use the INSERT and DELETE keys to change its length.

#### Defining Data Fields

When the format generator creates a data field, it defines it as a general field with no special attributes or characteristics. If you want to change the field definition, use the Field Definition menu. To display this menu, position the cursor on any part of the field and press SELECT. Figure 14-3 shows the Field Definition menu.

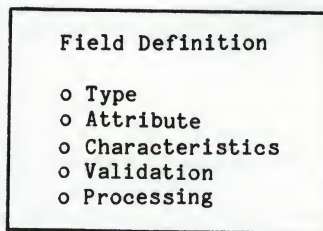


Figure 14-3. Field Definition Menu

Position the cursor on the field definition item you want to change and press ACCEPT. The format generator displays the corresponding submenu.

To select an option from a submenu, position the cursor on the option and press SELCT. If you select an option by mistake, you can deselect it by positioning the cursor on the option and pressing DESELECT. Press ACCEPT when you complete a submenu.

#### FIELD TYPE MENU

The Field Type menu allows you to change the field type. To display this menu, select the Type option of the Field Definition menu. Figure 14-4 shows the Field Type menu.

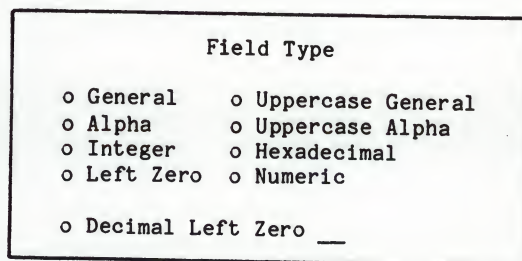


Figure 14-4. Field Type Menu

You can select only one field type. If you want to select the Decimal Left Zero option, you must enter the number of decimal places in the space next to the option. See Section 2, "Field Types," for details on field types.

#### FIELD ATTRIBUTES MENU

The Field Attributes menu allows you to select one or more field attributes. To display this menu, select the Attribute option of the Field Definition menu. Figure 14-5 shows the Field Attributes menu.

Field Attributes			
<input type="radio"/> High	<input type="radio"/> Underline		
<input type="radio"/> Low	<input type="radio"/> Flashing		
<input type="radio"/> Reverse	<input type="radio"/> Blank		
 <input type="radio"/> Character Color _			
1	2	3	4
5	6	7	8
 <input type="radio"/> Background Color _			
1	2	3	4
5	6	7	8

Figure 14-5. Field Attributes Menu

If you want to select a Character Color or a Background Color, you must enter a color number. The numbered colors below the options show which colors are available. Section 4, "Attributes," explains field attributes.

#### FIELD CHARACTERISTICS MENU

The Field Characteristics menu allows you to select one or more field characteristics. To display this menu, select the Characteristics option of the Field Definition menu. Figure 14-6 shows the Field Characteristics menu.



Field Characteristics	
<input type="radio"/> Must Enter	<input type="radio"/> Right Justify
<input type="radio"/> Must Fill	<input type="radio"/> Display Only
<input type="radio"/> Must Release	

Figure 14-6. Field Characteristics Menu

Field characteristics correspond to the E, F, R, J, and Q modifiers explained in Section 3, "Field Modifiers."

#### FIELD VALIDATION MENU

The Field Validation menu allows you to specify several different types of field validity checks. To display this menu, select the Validation option of the Field Definition menu. Figure 14-7 shows the Field Validation Menu.

Field Validation	
<input type="radio"/> Range Check	
Lower Limit:	_____
Upper Limit:	_____
<input type="radio"/> Check Digit:	__
<input type="radio"/> Value Set:	__
<input type="radio"/> Flag For Verification	

Figure 14-7. Field Validation Menu

If you want to select the Range Check option, you must enter the lower and upper limits. Range Check corresponds to the relational operators found in Section 8, "Arithmetic and Logic."



The Check Digit option corresponds to the C field modifier described in Section 3, "Field Modifiers." You must enter the modulus number before you select Check Digit. The format generator uses the length of the field as the check digit length parameter.

If you want to select the Value Set option, you must first enter the V (value set) descriptor. Section 9, "Value Sets," discusses the V descriptor.

The Flag For Verification option uses the V (verify on) and the V# (verify off) commands to flag the field for key verification. Section 7, "Other Commands," explains the V and the V# commands.

#### FIELD PROCESSING MENU

The Field Processing menu allows you to specify field processing. To display this menu, select the Processing option of the Field Definition menu. Figure 14-8 shows the Field Processing menu.

Field Processing

- o Add To Accumulator:     \_\_\_
- o Subtract From Accumulator:     \_\_\_
- o Execute Function Number:     \_\_\_
  
- o Include Code From Job,Batch  
\_\_\_\_\_
  
- o Branch To Job,Batch  
\_\_\_\_\_

Figure 14-8. Field Processing Menu

The Add, Subtract, and Execute options correspond to the B, Z, and X modifiers described in Section 3, "Field Modifiers." The Include option corresponds to the INCLUDE command and the Branch option corresponds to the BRANCH command. See Section 7, "Other Commands" for information on the INCLUDE and the BRANCH commands.

When you select any of the options in this menu, you must enter the appropriate value in the space next to the option. If you are not sure what to enter, refer to the appropriate section of this manual.

### COMPILING THE CODE

To compile and check your code:

- 1 Press EXIT. If you are writing into a batch that doesn't contain source code, LiFE-Works writes to the batch and exits the format generator.

If you are writing into a batch that contains source code, the format generator displays the menu shown in Figure 14-9.

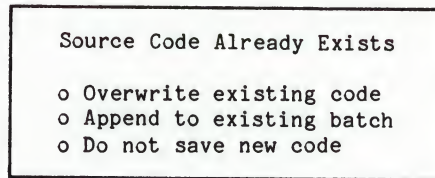


Figure 14-9. Source Code Already Exists Menu

Position the cursor on the option you want and press ACCEPT. LiFE-Works performs that option and exits the format generator.

- 2 If you want to review your format code before you compile it, you can use any of the following methods:
  - Use mode E to review your code one record at a time.
  - Use mode M to review your program one screen at a time. See Appendix C for information on the mode M editor.
  - Select mode I to access the vi editor. See the System V/68 User's Guide for instructions on using vi.
- 3 Compile your format code. Section 1 contains instructions on compiling a format.









## Appendix A Record Processing

### INTRODUCTION

This appendix, which describes the internal processing of LIfE-Works, provides information that can be useful in writing very complex format programs.

There are three passes through a record when it is processed with a format program: pass 1 (display), pass 2 (execution), and pass 3 (update).

### Pass 1 (Display)

Pass 1 occurs when a request is made to display an existing record. For example, pass 1 is used when paging through a batch in find or verify modes, or in entry mode when the UP ARROW and DOWN ARROW keys are used.

The following functions are performed during Pass 1:

- The record is moved from the disk to the screen according to the field type specifications (A, a, G, g, H, I, L, N) indicated in the format code. Note, however, that no field type validation is performed.
- Prompt characters, such as P, B, or @, are extracted from the format code and copied to the screen as "protected characters."
- ERASE and BLANK commands (Ew, Er,c, Bw, and Br,c) are executed.
- Text fields are moved from the disk; text characters in the format code are not used.
- Validation and generation commands are turned off. This includes value sets, field scan, generates from previous fields, and the G modifier.
- All editing and key validation modifiers are ignored. These include the E, F, R, A, D, and S modifiers.
- All "add" functions (Ba modifiers) are subtracted out of the accumulators.
- All "subtract" functions (Za modifiers) are added to the accumulators.
- The logical expressions in IF statements are evaluated so that the proper branches are taken.
- SET statements and Za commands are executed.

Because validation is off during pass 1, care should be taken in coding SPLIT statements that are based on the contents of accumulator 0. This is particularly true when the branches of the SPLIT statement affect other accumulators. Consider the following example:

```
G2=V1
SPLIT 4 WAYS ON A0
    L7B3 (branch 0)
    L7B4 (branch 1)
    L7B5 (branch 2)
    L7B6 (branch 3)
```

Since validation is off during pass 1, the zero branch of this SPLIT statement will always be taken during pass 1. In contrast, in the following statements

```
G2
IF!V1 THEN "ERROR"
SPLIT 4 WAYS ON A0
    L7B3 (branch 0)
    L7B4 (branch 1)
    L7B5 (branch 2)
    L7B6 (branch 3)
```

the value set check is made because IF statements are evaluated during Pass 1. Since the value set check is made, accumulator 0 is set correctly for the SPLIT statement and the proper branch is taken.

### Pass 2 (Execution)

Pass 2 begins when a new record is started or when an existing record is changed or released unchanged.

The following functions are performed during pass 2:

- The format code is executed from its beginning. Erases are performed (on a new record only) and prompts are put on the screen. The cursor is positioned at the first column at which keying is to begin.
- The operator can move the cursor through fields on the screen.
- Validation and generation are in effect.
- Editing and key validation modifiers are checked.
- All logical and SET statements are executed as they are encountered in the format.
- IF statements are evaluated and the proper branches are taken.
- All numeric calculations are made.



### Pass 3 (Update)

Pass 3 is responsible for writing released records to disk; it does not involve a pass through the format code. Pass 3 occurs when a new record is released or when an existing record is changed.

### USING THE PASTWO CONDITIONAL INDICATOR

Format programs based on the use of the PASTWO conditional indicator can be written in special situations. Before coding such programs, check with your Customer Service Representative.

The PASTWO conditional indicator is true only when pass 2 record processing occurs. System processing of a complex format program can be enhanced by checking this indicator in an IF test at the start of the format. One branch of this conditional can contain the complete format program and the other branch can contain only those format program statements that are executed during pass 1. The resulting saving is that the system is required to handle only an abbreviated form of the format program during all pass 1 processing.

This technique is most productive with format programs in which extensive validation checks are made, since these checks can be eliminated from pass 1 coding. All field type modifiers except Ba and Za can also be eliminated from pass 1 coding. But IF tests and prompts should usually remain for pass 1 processing.

Thus, the entire format program has the following form:

```
IF PASTWO

THEN      (complete format program with all editing and validation checks)

ELSE      (abbreviated format with only those statements required for pass
1)
```

The following simple example illustrates the principles involved in structuring a format program based on a PASTWO test. Consider the display in Figure A-1 and the format program coded for it.

The complete format program is placed in the THEN clause so it is executed only when pass 2 occurs (Figure A-2).

The abbreviated format program is placed in the ELSE clause so that it can be executed on pass 1 or pass 3. Note that any character could be specified as a code by the T command since the text field is copied from disk, not generated. Note also that Ba (and also Za) modifiers must remain for pass 1 processing.



Appendix A  
Record Processing

PROGRAM		COMMENTS		PAGE 1 OF 1																																																																																	
PROGRAMMER				DATE 12-18-85																																																																																	
ROW	DCAL ADDRESS	COLUMN																																																																																			
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81			
		DCAL ADDRESS																																																																																			
		00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81		
VIDE AREA																																																																																					
1	340																																																																																				
2	350	ACCT #: XXXXXXXX CODE: X AMT: XXXXXXXX																																																																																			
3	360																																																																																				
4	370																																																																																				
5	380	DATE: XXX OP: XXX																																																																																			
6	390																																																																																				
7	400																																																																																				
8	410																																																																																				
9	420																																																																																				
10	430																																																																																				
11	440																																																																																				
12	450																																																																																				

A5138A

Figure A-1. Sample Screen Display

```

IF PASTWO
THEN (B2,1 P"ACCT #:" E432 G7EC6,10 P" CODE:"
      (IF>"5"<"0" THEN T"1" ELSE (IF>"7"<"4" THEN T"2" ELSE T"3"))
      P" AMT:" L8<"18"B3 B4,1 P"DATE:" L3E
      (IF<"366">"000" THEN "JULIAN DATE 001-365")
      P" OP:" G3G#OPER)

ELSE (B2,1 P"ACCT #:" E432 G7 P" CODE:" T"1"
      P" AMT:" L8B3 B4,1 P"DATE:" L3
      P" OP:" G3)

```

Figure A-2. PASTWO Format Code

### APPLICATION EXAMPLES

The following discussion begins with two application examples, one using batch balancing and the other using index set updates. Both of these examples have special requirements when existing records are displayed or changed. After these applications are discussed, a description of how pass 1/pass 2 logic was implemented is presented. Lastly, there is a discussion of problems that may occur if pass 1/pass 2 is not taken into consideration when format code is written.

#### Example 1: Batch Balancing

A batch consists of five checks with values \$50, \$45, \$60, \$15, and \$10. The operator enters these checks as \$50, \$44, \$60, \$15, and \$10 (\$44 instead of \$45). The correct batch balance total (that is, the total of all checks in the batch) for these checks is \$180; however, the batch balance total for the amount entered by the operator is \$179. Therefore, the batch is out of balance by \$1.

LiFE-Works has many ways to verify that data has been entered correctly. We will look at one of those that can be used to "catch" errors such as the one described above. Consider the following format code:

```
$FORMAT 702 $SIZE 7  
P "ENTER BATCH TOTAL:" L7B1 F2  
$FORMAT 703 $SIZE 6  
P "ENTER CHECK AMOUNT: " L6Z1
```

If a job is defined with these two program levels, it can be used to enter a batch control total which is used to verify the rest of the data. The job is defined with one accumulator, A1; that accumulator is the batch balance accumulator for the job. (LiFE-Works considers the highest numbered accumulator to be the batch balance accumulator and treats it somewhat differently from other accumulators.) The job has the following definition:

```
$DEFJOB CHECKS $FORMAT 702,703 $MAXSIZ 7 $ACCUMS 1
```

The operator enters the data described above into CHECKS,1. The first screen displays the following prompt (Record 1)

```
Record 1:      ENTER BATCH TOTAL:
```



Appendix A  
Record Processing

The operator enters the batch control total, \$180, in the L7B1 field. The B1 modifier tells the system to add the contents of the entered field into accumulator 1. Therefore, A1 now contains 180 (all accumulators are initialized to zero when each new batch is entered). The operator then enters the check amounts in the L6Z1 field:

Record 2:	ENTER CHECK AMOUNT: 0000050
Record 3:	ENTER CHECK AMOUNT: 0000044
Record 4:	ENTER CHECK AMOUNT: 0000060
Record 5:	ENTER CHECK AMOUNT: 0000015
Record 6:	ENTER CHECK AMOUNT: 0000010

Each time the operator enters an amount in the L6Z1 field, the Z1 modifier tells the system to subtract the amount from accumulator 1. The table below shows the value contained in accumulator 1 after each entry:

Record Number	Amount Entered	Resulting Contents of Accumulator 1
1	180	000000000180
2	50	000000000130 (180-50 = 130)
3	44	000000000086 (130-44 = 86)
4	60	000000000026 (86-60 = 26)
5	15	000000000011 (26-15 = 11)
6	10	000000000001 (11-10 = 1)

At this point, the operator exits the batch. Since A1, the batch balance accumulator for this job, now contains 1 (instead of zero), the batch is considered out of balance. (Any time a batch belonging to a job that uses accumulators has been entered, the system checks the highest-numbered accumulator. If that accumulator is zero, the batch is considered in balance and the INBAL flag in the batch directory for that job and batch is set true. If the highest-numbered accumulator contains any value other than zero, the batch is considered out of balance and the INBAL flag is set false.)

The operator goes back through the records to correct the mistake and notices that record 3 contains 44 instead of 45. The difference between the entered value and the correct value is one, which is the amount by which the batch is out of balance. The operator opens the record for correction and enters 45 in that field. Since the field is coded L6Z1, that amount is added to the accumulator. But 44 has to be subtracted before the correct value, 45, can be entered, or the accumulator does not come out to zero. Without pass 1 and pass 2, entering the correct amount at this point would not balance the batch, because the incorrect value is still in there. To get around this, pass 1 automatically subtracted 44 from the accumulator when the record was displayed, in anticipation of corrections. If no corrections were made, 44 would be added back in again. When a correction was made, the new amount went into the accumulator, and the accumulator is now in balance. See "Effect on Batch Balancing" below for a step-by-step description of the effects of pass 1 and 2 on this example.

### Example 2: Index Set Update

This example uses an index set that is being updated with a SET command. Assume that the index set contains payroll information and is being updated to contain year-to-date amounts. Part of the code might look like this:

```
P"GROSS PAY: " 1L8.2E
SET X(125,3,8)=X(125,3,8)+F1
```

The operator must first enter a key (such as the employee number) which is used to select the desired record in the index set. After the record is selected, it can be updated. In the code shown above, the year-to-date gross pay field is updated with the gross pay for the payroll period.

The gross pay of each of the employees listed below is used to update the year-to-date gross pay field in the index set record.

Employee Number	Gross Pay	Year to Date Gross Pay	
		Before	After
2501	1000	3000	4000
7403	1100	3300	4400
7406	800	2400	3200

The operator makes a mistake in entering the gross pay for employee #7403 and enters \$1000 instead of \$1100. This makes the Year-to-Date gross pay \$4300 instead of \$4400. To correct this, the operator has to redisplay the record for employee #7403 and open it for correction, then replace the amount \$1000 (for the gross pay) \$1100. What about the index set record, which still contains \$4300 as the year-to-date gross pay? If the SET statement is executed again, the year-to-date figure will become \$5400 (instead of \$4400). How can this be avoided? The original value has to be subtracted out of the index set before the operator corrects the record. Pass 1 does not automatically back out the incorrect value of a SET statement, unlike the batch balancing commands in the first example. This is the programmer's responsibility. See "Effect on Index Set Update" below for instructions.

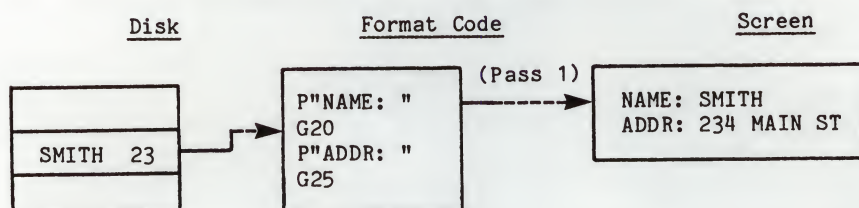
### Implementation of Pass 1, Pass 2, and Pass 3

The above considerations regarding batch balancing and index set updates lead to the implementation of a feature called pass 1/pass 2. Pass 1 and pass 2 refer to different ways the format code is executed. When format code is executed in Pass 2 mode, it works the way the manuals say it does. For example, P'TEXT' would cause the characters TEXT to be displayed on the screen and not inserted into the output record; L7B1 would cause the amount entered to be added to accumulator one. Pass 2 mode is the way format code is executed when a new record is entered.



However, problems can occur if changes are made to existing records such as the ones in examples 1 and 2. To compensate, pass 1 mode is used. In pass 1 mode format code executes in a way that differs from the usual way (that is, pass 2). This happens so that records can be corrected so that accumulators and index set records contain the correct values.

First, consider what happens when an existing record is displayed on the screen. The record is stored on the disk without any prompts or spaces; only the actual data is on the disk. To display the record on the screen, the system must use the format code to determine the screen layout. The format code supplies the prompts and spaces that are omitted from the disk record; it also supplies the correct logic paths from the IF/THEN and SPLIT statements. The following diagram may help you to visualize this:



In anticipation of corrections to fields used in batch balancing, the system executes B and Z modifiers in reverse. That is, if there is a command L7B1 in the format code, the system takes the next seven characters from the disk record and displays it on the screen. Since the command has a B modifier and it is displaying an existing record on the screen, it subtracts the amount from rather than add it to accumulator one. If the command had been L7Z1, the system would have taken the next seven characters from the disk record, displayed them on the screen, and added them (the reverse of the usual way the Z modifier works) to accumulator one.

This mode of executing format code (that is, displaying an existing record on the screen) is called pass 1 mode. After Pass 1 mode displays the record on the screen, pass 2 mode is used to allow the operator to enter corrections if necessary. Pass 2 mode executes even if the operator does not enter corrections. This ensures that the system puts back into the accumulators the values backed out from them during pass 1.

The third mode, pass 3, does not execute format code, unlike pass 1 and pass 2. Pass 3 is used to write records disk. It scans the screen for data characters and puts just those data characters into a disk record.

Information about the three modes can be summarized as follows:

Pass 1

- Occurs whenever an existing record is displayed.
- Executes format code to determine screen layout.
- Executes format code to determine prompts.
- Retrieves data from the disk record.
- Reverses execution of B and Z modifiers.

Pass 2

- Occurs whenever a new record is entered.
- Occurs after an existing record is displayed in Pass 1 mode.
- Occurs after an existing record is released (feed-through mode).

Pass 3

- Extracts data characters from screen and creates a disk record.
- No format code is executed.

Relationship Between Pass 1, Pass 2, and Pass 3

Let us look at how these different passes are related. When a new record is entered, the format code is executed in pass 2 mode. When the record is released, pass 3 mode writes the record to disk, then pass 2 mode is used to enter the next record.

Figure A-3 illustrates the use of pass 2 and pass 3 modes in creating a new record.

Appendix A  
Record Processing

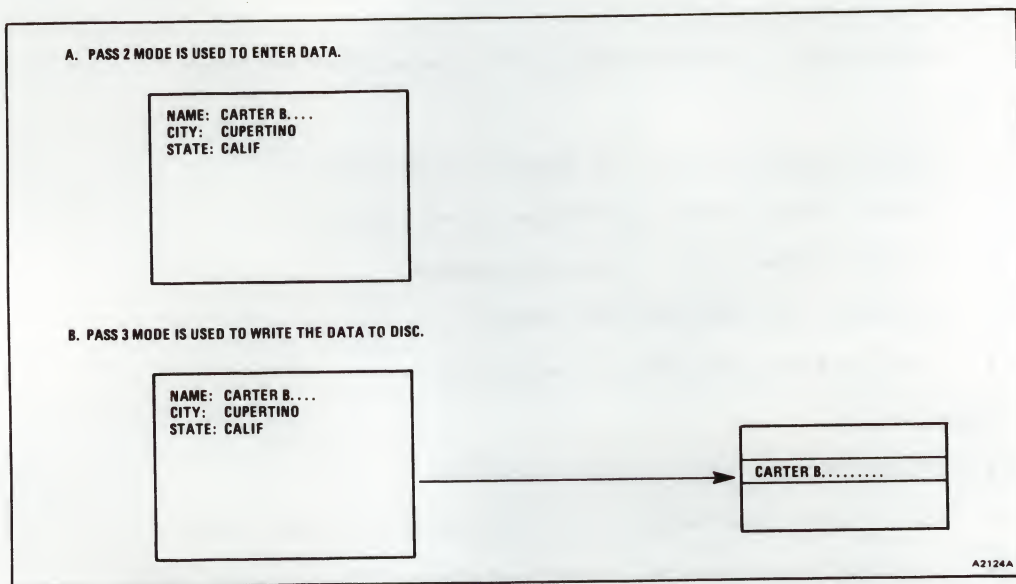


Figure A-3. Creating a New Record

Pass 1 and pass 2 are always executed if an existing record is displayed on the screen. Pass 3 mode is executed only if the record is opened for correction and modified. Let's see how these are related.

Pass 1 mode is used to display the record on the screen. Pass 2 mode is then executed up to the first field requiring input. For example, the format might begin with a prompt followed by a G25 command. Pass 2 would execute the prompt and pause with the cursor at the beginning of the G25 field. If the operator releases the record or presses the UP ARROW, DOWN ARROW, or EXIT keys, the remainder of the format code is executed in Pass 2 (feed-through) mode. (The system always "finishes" the record.) Pass 3 mode is not executed since the record was not opened for correction.

If instead the operator opens the record for correction, the operator can make changes to the record. In making the changes, the system executes the format code in Pass 2 mode. When the record is released, if not at the end of the record, the remainder of the format will be executed in Pass 2 (feed-through) mode. After the entire format has been executed, pass 3 mode causes the record to be rewritten.

These situations are illustrated in Figure A-4.



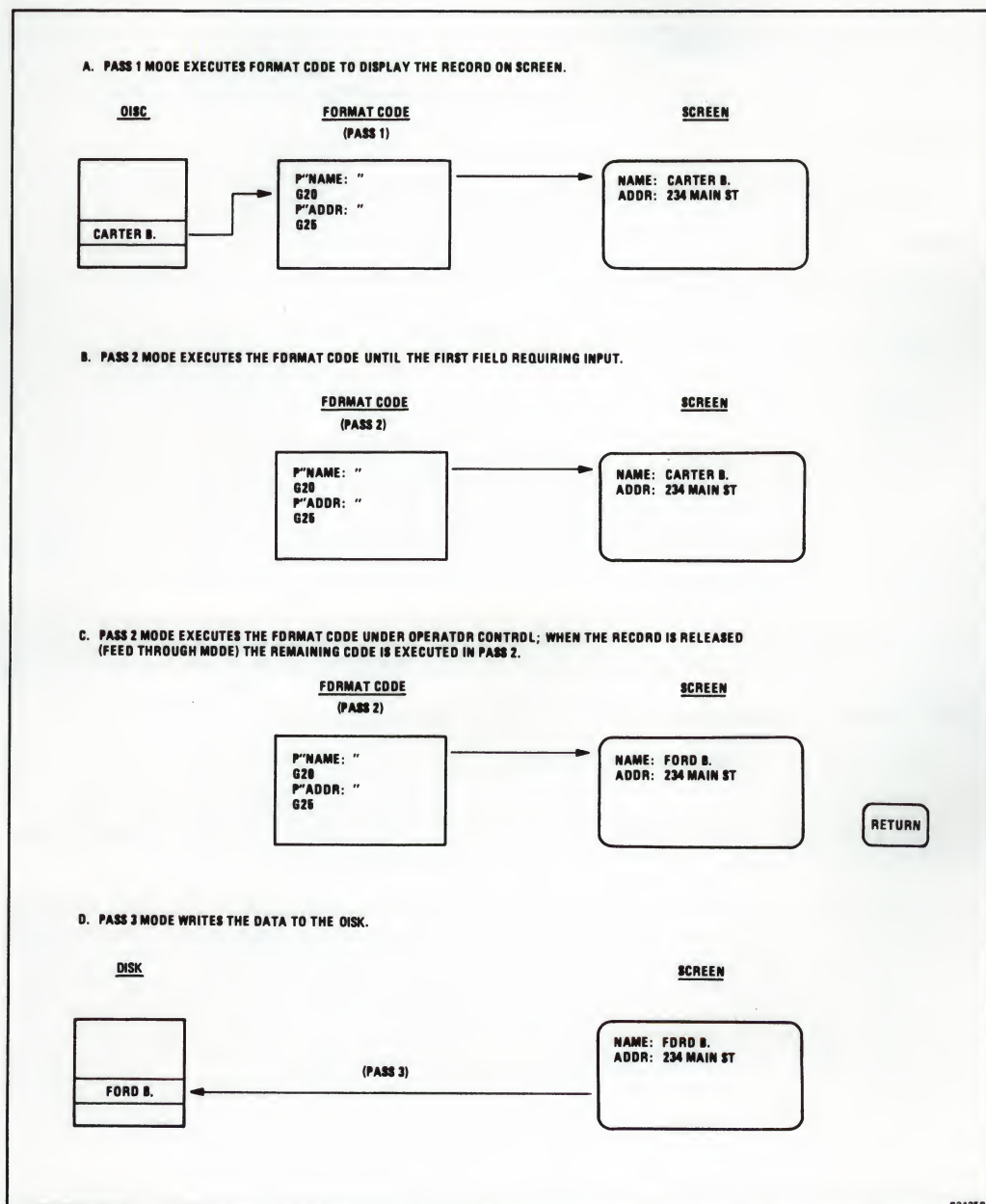


Figure A-4. Modifying an Existing Record



Effect on Batch Balancing

The earlier example of batch balancing used two levels of format code:

Program Level 1:

P"ENTER BATCH TOTAL:" L7B1 F2

Program Level 2:

P"ENTER CHECK AMOUNT:" L6Z1

The operator had entered the following data:

Record Number	Amount
1	\$180
2	\$ 50
3	\$ 44 (should have been \$45)
4	\$ 60
5	\$ 15
6	\$ 10

After all the data has been entered, A1 (the batch balance accumulator) contains 1 (out of balance).

After record 6 has been released, record 7 displays.

Record 7:        ENTER CHECK AMOUNT:

The cursor is positioned at the beginning of the field to be updated. To display record 3 on the screen for correction, the operator must press the UP ARROW key four times.

The first time UP ARROW is pressed, the system executes pass 1 mode to display record 6 on the screen. During this pass through the format code, the modifier in the command L6Z1 is executed in reverse; that is, the system retrieves six data characters from the disk (000010), displays those on the screen, and adds that amount to A1 in anticipation of corrections. Then pass 2 takes over and begins executing the format code up to the L6Z1 which is the first field requiring input; the system pauses there with the cursor at the beginning of the field.

These steps are summarized in the following diagrams.

A1 = 1

ENTER CHECK AMOUNT:
7

UP ARROW

results in the following:

Pass 1

A1 = 11

ENTER CHECK AMOUNT: 000010
6

Pass 1 has been executed for record 6. The system reads six characters from Record 6 (on disk) and displays them on the screen. Since the field has a Z modifier, pass 1 executes this in reverse; therefore, it adds the amount to A1 resulting in the contents of the accumulator being 11. Then pass 2 begins executing and pauses at the beginning of the first field requiring input.

Pass 2

A1 = 11

ENTER CHECK AMOUNT: 000010
6

Since record 6 is not the record to be updated, the process must be repeated.

A1 = 11

ENTER CHECK AMOUNT: 000010
6

UP ARROW

Appendix A  
Record Processing

First, pass 2 must be completed for the displayed record since it is an existing record. This means the L6Z1 is executed in the usual way; therefore, 10 is subtracted from A1.

Pass 2

A1 = 1

ENTER CHECK AMOUNT: 000010
6

Next, pass 1 is executed for record 5. Pass 1 adds 15 to A1.

Pass 1

A1 = 16

ENTER CHECK AMOUNT: 000015
5

Then pass 2 is executed up to the first field requiring input. Since record 5 is not the record to be updated, the process must be repeated.

Pass 2

A1 = 16

ENTER CHECK AMOUNT: 000015
6

UP ARROW

First, pass 2 must be completed for the displayed record since it's an existing record. Pass 2 executes L6Z1 in the usual way; therefore, 15 is subtracted from A1.

Pass 2

A1 = 1

ENTER CHECK AMOUNT: 000015
5

Next, pass 1 is executed for record 4. Pass 1 adds 60 to A1.

Pass 1

A1 = 61

ENTER CHECK AMOUNT: 000060
4

Then pass 2 is executed up to the first field requiring input. Since record 4 is not the record to be updated, the process must be repeated.

Pass 2

A1 = 61

ENTER CHECK AMOUNT: 000060
4

UP ARROW

First, pass 2 must be completed for the displayed record since it is an existing record. Pass 2 executes L6Z1 in the usual way; therefore, 60 is subtracted from A1.

Pass 2

A1 = 1

ENTER CHECK AMOUNT: 000060
4

Next, pass 1 is executed for record 3. Pass 1 adds 44 to A1.

Pass 1

A1 = 45

ENTER CHECK AMOUNT: 000044
3



Appendix A  
Record Processing

Then pass 2 is executed up to the first field requiring input.

Pass 2

A1 = 45

ENTER CHECK AMOUNT: 000044
3

CORR

Since record 3 is the record to be corrected, the operator presses CORR to open the record for correction. The correct amount, 45, is then entered. Since the record is now executing in pass 2 mode, that amount is subtracted from A1 resulting in the batch balance accumulator containing zero (the batch is in balance). After the field has been entered, pass 3 mode is used to write the record out to disk, replacing the incorrect record with the correct one.

Pass 2

A1 = 0

ENTER CHECK AMOUNT: 000045
3

As soon as record 3 has been written to the disk, the system displays the next record (4) on the screen and adds the amount 60 into A1; it then begins pass 2. At this point the operator presses MODE to close the batch, and record 4 finishes executing pass 2 so that A1 now contains zero. The batch is now in balance. (Unless the system fails, LIFE-Works always completely executes pass 2 mode if it executed a pass 1 mode.)

Therefore, pass 1 and pass 2 handle the B and Z modifiers to allow for the operator's corrections. Pass 1 backs out of the accumulator the original value and pass 2 adds the new or existing value back into an accumulator.

### Effect on Index Set Update

Using the SET command to update an index set poses a slightly different problem for LiFE-Works. Since the SET command can have almost any kind of expression in it, it would be extremely difficult for LiFE-Works to determine how to back out the "old" value in anticipation of a correction. For example:

SET X(125,3,8)=F1

SET X(125,3,8)=X(125,3,8)+F1

SET X(125,3,8)=X(125,3,8)+F1+F2-F3-F4

In the first example, a field in the index set is set to a single value. In the second example, a field in the index set is used to keep a sum of a particular field (this is how the year-to-date figures are kept). And in the last example, a more complicated formula is used. How could LiFE-Works devise a way to determine what should be backed out in pass 1?

The answer is that it is the programmer's responsibility to back out the necessary amount. To allow this, LiFE-Works provides two conditional indicators, NEWREC and PASTWO. NEWREC is true only when a new record is being entered. PASTWO is true only while pass 2 is executing; therefore, it is false when pass 1 is executing.

Different combinations of NEWREC and PASTWO are summarized in Table A-1.

Table A-1. Combinations of NEWREC and PASTWO

Condition	TRUE	FALSE
NEWREC	Only when entering a new record	Record is an existing record
PASTWO	Record is a new record or an old record executing Pass 2	Pass 1 mode for an existing record

We can make use of these conditional indicators to make sure pass 1 and pass 2 properly handle updates to index sets. Consider the following statement.

IF PASTWO THEN (SET X(125,3,8)=X(125,3,8)+F1)

ELSE (SET X(125,3,8)=X(125,3,8)-F1)

Appendix A  
Record Processing

This statement can be used to ensure that, if corrections are made to a record, the index set reflects the correct value. This statement updates the index set when the record is first entered. When an existing record is displayed, pass 1 backs out the value that was previously entered in anticipation of a correction. When pass 2 is executed, the new value entered into field 1, if field 1 were updated, or the old value if field 1 were not changed, would be added back into the index set.

When new records are being entered, the field entered is used to update the corresponding field in the index set. When an existing record displays, it displays in Pass 1 mode, which causes the system to back out of the index set the value for that field. This is analogous to what Pass 1 does with the B and Z modifiers. Therefore, if the operator enters a new value, that value (i.e., the correct one) is reflected in the index set. If the operator does not enter a new value, the old value is used to update the index set. In either case, the index set is updated correctly.

You may have inferred from this that the SET command is executed both in pass 1 mode and in pass 2 mode. That is true for the SET command, but not for all other commands. (For example, you will notice in the next example that the GA1 in the 2I3GA1 command is executed when a new record is entered provided that ASD is on, is not executed in pass 1 mode, and is not executed in pass 2 mode for an existing record in feed-through mode.) As a result of the execution of the SET command in pass 1 and pass 2 mode, care must be exercised when the SET command is used.

Consider the following example where a SET command is used to generate a record number, which is then stored in the data record.

```
E400
SET A1=A1+1
P"NAME:" 1G20E
B2,1 ASD P"RECD:" 2I3GA1
```

As new records are being entered, A1 is incremented in such a way that the record number field will contain 001 for the first record, 002 for the second, and so on. Assume the following data is entered:

Record	Field 1	Field 2
1	JONES	001
2	SMHTH (should be SMITH)	002
3	LANE	003

Where, field 2 is the record number field and is automatically generated from the contents of A1. Assume that the name SMHTH in Record 2 should have been entered as SMITH and the operator wants to go back to Record 2 to correct it. After making the correction, the operator wants to then continue entering records.



After record 3 has been entered, the system executes pass 3 mode to write the record to the disk and then begins executing record 4 in pass 2 mode up to the first field requiring input. This means that the erase, SET, and prompt commands are executed; therefore, A1 contains 4. Since the operator wants to correct record 2, the operator must press the UP ARROW key twice to display that record.

Pass 2

A1 = 4

NAME:
4

UP ARROW

The remainder of Record 4 is ignored and pass 1 is used to display record 3 on the screen. Remember, the SET command at the beginning of the format is executed in pass 1 mode; therefore, A1 contains 5 after pass 1 displays the record.

Pass 1

A1 = 5

NAME: LANE
RECD: 003
3

Then pass 2 takes over and executes the format code up to the first field requiring input. Pass 2 executes the SET command again.

Pass 2

A1 = 6

NAME: LANE
RECD: 003
3



Appendix A  
Record Processing

Since record 3 is not the desired record, the UP ARROW key is pressed again and the remainder of the format code for record 3 is executed in pass 2.

Pass 2

A1 = 6

NAME: LANE  
RECD: 003

3

UP ARROW

Then record 2 is displayed using pass 1 mode; again, A1 is incremented.

Pass 1

A1 = 7

NAME: SMHTH  
RECD: 002

2

And, pass 2 executes the code up to the first field requiring input. Pass 2 executes the SET command again!

Pass 2

A1 = 8

NAME: SMHTH  
RECD: 002

2

At this point, the operator presses CORR, corrects the name, then releases the record.

Pass 2

A1 = 8

NAME: SMITH  
RECD: 002

2

DOWN ARROW

The remainder of the format code is executed in pass 2 mode. Then, since the record was opened for correction, pass 3 is executed and the record is written to the disk.

Then, pass 1 mode is executed to display record 3 on the screen, after which, pass 2 executes up to the first field requiring input. Thus, the SET command executes two more times. The operator presses DOWN ARROW to reach record 4.

A1 = 10

NAME: LANE RECD: 003          3
--

DOWN ARROW

Finally, record 4 can be entered. (Remember, pass 2 executes up to the first field requiring input, so the SET command is executed again!)

Pass 2

A1 = 11

NAME:          4
--

After the name has been entered, the format code generates the record number field from accumulator one. Therefore, record 4 contains 11 as its record number field.

A Sample Problem

This last example illustrates a problem that can occur if SET An commands are used in format code in such a way that the values of fields in the data record are generated from the accumulator. PASTWO logic in the format code might solve the problem. Consider the following.

```
E400
IF PASTWO THEN SET A1=A1+1
           ELSE SET A1=A1-1
P"NAME:" 1G20E
B2,1 ASD P"RECD:" 2I3GA1
```

Does this coding solve the problem? Try it first on a sheet of paper and then on the system. Determine if the record number field is correctly generated even if the operator presses UP ARROW, corrects an existing record, or inserts a record. (To check this, enter the code into a LiFE-Works system and enter several records. After entering several records, page back to the first record and then page forward to the last record entered. Then enter a new record.)

### Discussion of Sample Problem

The code just given for the sample problem on the previous page is used to generate a sequential record number that is stored as part of the data record:

```

E400
IF PASTWO THEN SET A1=A1+1
           ELSE SET A1=A1-1
P"NAME:" 1G20E
B2,1 ASD P"RECD:" 2I3GA1

```

The above code does not work correctly if the operator uses the UP ARROW key. The reason for this is that the IF statement containing the SET command precedes the first field requiring input. Therefore, as each new record is displayed, the SET command that increments the accumulator is executed before the operator is given a chance to enter anything, even the UP ARROW key.

Assume (as in the last example of the previous section) that the operator enters three records. When the fourth record is displayed, the operator presses the UP ARROW key to redisplay the third record.

Pass 2
$$A1 = 4$$

NAME :		4	UP ARROW
--------	--	---	----------

The remainder of record 4 is ignored, and pass 1 is used to display record 3 on the screen. The IF statement is false, so the ELSE-branch is executed; this causes the accumulator to be decremented.

Pass 1

**A1 = 3**

NAME: LANE  
RECD: 003



Then pass 2 takes over and executes the format code up to the first field requiring input. Pass 2 executes the IF statement again; since PASTWO is now true, the THEN-branch is executed, which causes the accumulator to be incremented.

Pass 2

A1 = 4

NAME: LANE  
RECD: 003

3

Now, if when record 4 is displayed, the operator changes this record or just releases it, the IF statement at the beginning of the format code causes the accumulator to be incremented again.

Pass 2

A1 = 5

NAME:

4

Therefore, after the operator enters the name field, the record number field is generated from A1, so the record number field contains 005 instead of the correct value 004.

From this description, you might assume that the problem would be corrected simply by moving the IF statement so that it follows the first field requiring input:

```
E400
P"NAME:" 1G20E
B2,1 ASD
P"RECD:"
IF PASTWO THEN SET A1=A1+1
          ELSE SET A1=A1-1
2I3GA1
```

The above code works correctly even if the operator uses UP ARROW to back up to an existing record. This is a simplified example of a general rule that should be followed when writing format code. In general, remember this rule: if possible, place SET statements after all fields requiring operator input.



If the SET statements appear after all fields requiring operator input, the SET statements are not executed until after the operator has entered all fields requiring input. Therefore, if an operator decides to use the UP ARROW key to display an earlier record while entering a record, the accumulators have not been incremented yet and should contain the correct value when the new record is finally entered.

#### Other Problems

There is another problem with SET statements. The problem occurs if an operator backs up through format code, and happens to back up through a SET command. Consider the following example:

```
1G20 ASD
IF PASTWO THEN SET A1=A1+1
           ELSE SET A1=A1-1
2I3GA1 3G10 4I5
```

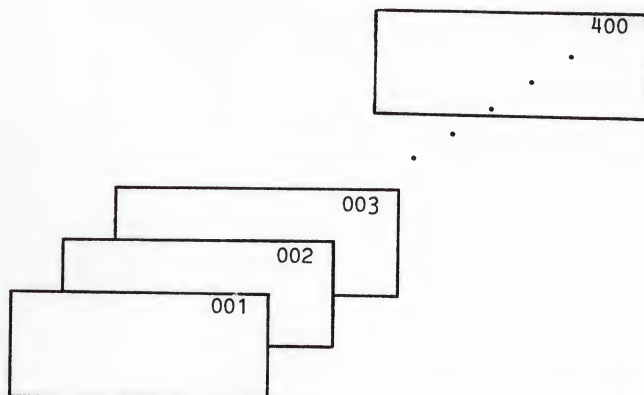
Assume that an operator is entering a new record and is entering field 3. At this point, the accumulator A1 has already been incremented. If the operator decides to back up through the format code, the cursor will be positioned at the beginning of field 1. In doing this, the IF-THEN-ELSE statement is skipped (that is, it will not be executed when it is backed through). However, after the operator enters field 1, the IF statement is executed again. As a result, the value of A1 will be one greater than it should be. (Note: if an operator backs through commands containing B and Z modifiers, the B and Z modifiers are executed in reverse. This action ensures that the current value is backed out in anticipation of its being entered again.)

This last example illustrates problems that occur when accumulators are used to generate field values. The use of accumulators as shown in this example should be avoided whenever possible, to prevent complications that arise if an operator backs through format code or displays existing records. For example, the keyword #RECNO should be used to generate the record number field.

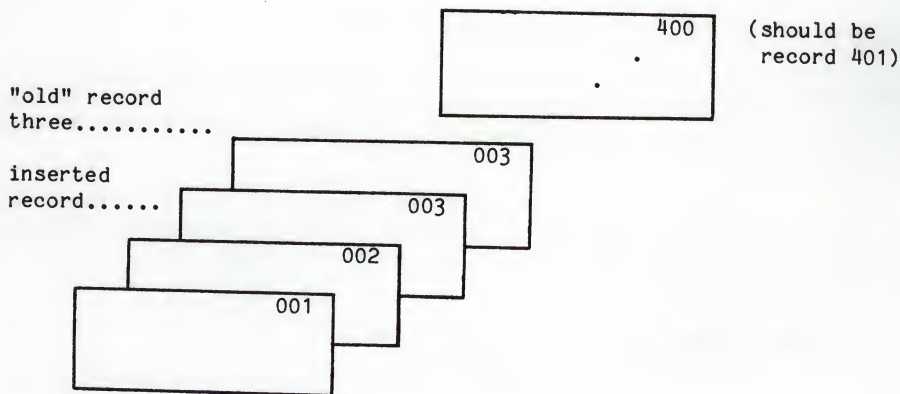
```
E400
P"NAME:" 1G20E
B2,1 ASD P"RECD:" 2I3G#RECNO
```

The above format code correctly generates the record number field even if the operator backs up through the format code or displays an existing record for correction. However, it does not work correctly if a record is inserted in a batch of existing records.

Assume that 400 records have been entered into a batch using the above format code. Each record would have a three position field containing the number of that record.

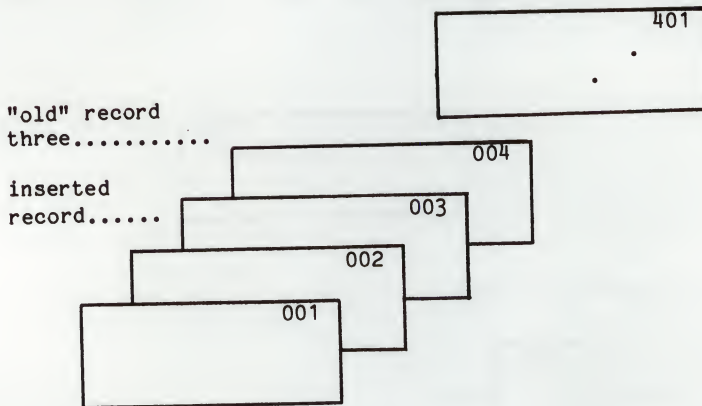


If the operator inserts a record after record two, the record number field is not correct for all records following the inserted record.



Verify Reconstruct is a LiFE-Works feature that can be used to correct errors such as the one described above. Verify Reconstruct recreates an entire batch. It requires no operator input since it takes the input data from the batch that had been entered. If Verify Reconstruct were run on the batch described above, the individual records of the resulting batch would have the correct value in the record number field.

Appendix A  
Record Processing



The following are performed during verify reconstruct:

- Each record is executed first in pass 1 and then in pass 2.
- All accumulator commands are executed and the resulting totals are used to determine if the batch is in balance.
- Validation tests are performed.
- The HAS BEEN VERIFIED flag for each record is cleared.

Because pass 1 and pass 2 are executed for each record, a new keyword is required. The keyword VRECON is true if verify reconstruct is being run; otherwise it is false. This technique can be used to ensure that accumulators and index sets are accurately reconstructed.

The following example shows how the VRECON keyword is used to restore an index set. Assume that the index set is restored from a backup; after it has been restored, all batches entered after the backup was taken should be run through in Verify Reconstruct mode.

```
IF PASTWO THEN SET X(125,3,8)=X(125,3,8)+F10
ELSE IF VRECON THEN NULL
ELSE SET X(125,3,8)=(125,3,8)-F10
```







Appendix B  
Check Digit Calculation

INTRODUCTION

A check digit is a system-calculated number added as a suffix to a base number. Check digits are used when the risk of error must be reduced to a minimum. When the operator enters a number with a check digit, the processor performs a modulus calculation to determine what the check digit should be. If the digit the operator enters and the one the processor calculates are different, there is a data entry or source document error.

A typical application of this system is a charge account number assignment. Assume that there is a list of five-digit numbers to be assigned to new accounts. The system processes this list using modulus calculation and adds a sixth digit, the check digit, to the five-digit base number. The new six-digit number then becomes the account number. If, during data entry, an operator makes a transposition or other error, the system detects it when it calculates for the sixth digit and finds it doesn't agree with the one entered.

LIF-Works uses moduli 7, 10, and 11. The C command generates a check digit, and the C modifier validates the check digit. The following paragraphs summarize the use of modulus checks and describe the methods by which the system calculates the various modulus checks. (See Section 7, "Other Commands," for a description of the C command; see Section 3, "Field Modifiers," for a description of the C modifier.)

Appendix B  
Check Digit Calculation

MODULUS 7

Modulus 7 detects most single transpositions and incorrect digits. The check digit is obtained by dividing the base number by 7 and placing the remainder to the right of the units digit (the right-most digit of the number). For example,

Basic number:                   9 4 3 1 2

9 4 3 1 2  $\div$  7 = 13473 with a remainder of 1

Self-checking number:       9 4 3 1 2 1

MODULUS 10

Modulus 10 detects single transpositions and incorrect digits. The modulus 10 check digit is calculated as follows:

- 1 Multiply every other digit by two, starting with the right-most digit. (The rightmost digit is always considered an odd digit.)
- 2 Add the digits produced by 1 to the even digits. (The even digits start with the second digit from the right.)
- 3 Subtract the total from 2 from the next higher multiple of 10.

For example:

Base number	9 6 4 3 8
Odd digits of base number	9 4 8
Multiply by 2	X 2
	<hr/>
	18 8 16
Even digits	6 3
Add	1+8+6+8+3+1+6 = 33
Next higher multiple of ten	40
Subtract total	-33
Check Digit	7
Self-checking number	9 5 4 3 8 7



Appendix B  
Check Digit Calculation

MODULUS 11

Modulus 11 detects incorrect digits, single transpositions, and double transpositions. It is different from other check-digit systems in that it is based on a weighted checking factor for each digit in the base number. The modulus 11 check digit is calculated as follows:

- 1 Each digit position of the base number is assigned a weighted checking factor. These factors are 2, 3, 4, 5, 6, 7; 2, 3, 4, 5 and so on starting with the units digit (right-most digit) and progressing toward the high-order digit (left-most digit).
- 2 Each digit in the base number is multiplied by the checking factor.
- 3 The products are added and divided by 11. The resulting remainder is subtracted from 11, which produces the check digit.

For example:

Basic number	5 1 6 1 9 2 8 7 2
Checking factors	4 3 2 7 6 5 4 3 2
Add the products	$20+3+12+7+54+10+32+21+4 = 163$
Divide by 11	$163 \div 11 = 14$ with a remainder of 9
Subtract remainder from 11	$11-9 = 2$ , the check digit
Self-checking number	5 1 6 1 9 2 8 7 2 2

Base numbers that require a check digit of 10 cannot be used as a self-check in modulus 11 operations. These numbers are defined as those whose products derived as in the above example are divisible by 11 with a remainder of 1. The next step would produce  $11-1=10$ , which is not permissible. If LIFE-Works is used to generate a check digit for one of these prohibited numbers, the message CHECK DIGIT ERROR flashes. When one of these numbers is in a continuous series of numbers entered for check digit generation, the numbers must be skipped, which leaves gaps in the series (every eleventh number).

Those products that are evenly divisible by 11 (no remainder) produce a check digit of 11 ( $11-0=11$ ). In these instances, the digit 0 is appended to the base number to produce a self-checking number. Only those product totals that are evenly divisible by 11 produce a check digit of 0.

### ALPHABETIC CHECK DIGITS

Strings of alphabetic characters or combinations of alphabetic and numeric characters can also be assigned check digits. Note that even on an alphabetic string, the check digit is never an alphabetic character. For example, the statements A7 C8,7 generates an integer check digit and appends it to the seven-character alphabetic field. When this field is checked during entry, the format code must allow an integer to be keyed in the last position; therefore, the field should be coded with the statements A7 I(-)1C8,7 or G8C8,7.

Any of the three moduli can be used to generate and check the check digit. The alphabetic characters are accommodated in the calculation by assigning them a numeric value as follows:

A B C D E F G H I J K L M

1 2 3 4 5 6 7 8 9 1 2 3 4

N O P Q R S T U V W X Y Z

5 6 7 8 9 2 3 4 5 6 7 8 9

The calculation treats each character as a number and derives the check digit as explained previously. Spaces are treated as zeros. An example using modulus 7 calculation follows:

Original part number: 3 4 B 1 2 H

B and H have a respective value of 2 and 8, so for the check digit calculation, the part number becomes 342128. Dividing 342128 by 7 yields 48875 with a remainder of 3. Therefore, the part number with the appended check digit becomes:

Self-checking number: 3 4 B 1 2 H 3









## Appendix C The LiFE-Works Editor

### INTRODUCTION

The mode M of LiFE-Works is a simple multirecord editor for entering and editing batches of format code and supervisory commands. The LiFE-Works editor lets you work with a screen full of records instead of one record at a time. It also provides a number of editing features. For example, it lets you switch between insert and overwrite modes, and duplicate or blank-fill a record with one keystroke.

To access the LiFE-Works editor, select mode M by pressing EXIT and M. Then enter the job and batch name. The job must already exist. If the batch does not exist, LiFE-Works asks if you want to create a new batch using that name.

Table C-1 describes the mode M function keys. Press HELP if you need to see how these keys relate to the keys on your keyboard.

### General Usage Notes

- Mode M displays records in sequence, with the first record in the batch at the top of the screen. Each record occupies one screen line.
- Mode M can be used on records of any length and on batches of any size. You can scroll the screen display left, right, up, and down to view all of the data in the batch. The display also scrolls automatically as needed; for example, if you reach the edge of the screen while typing, the display scrolls so you can continue typing.
- If the records are shorter than the screen width or if there are fewer records than screen lines, the unused areas of the screen are filled with a background pattern. You cannot move the cursor into an unused area.
- Mode M can be used only on fixed-length batches. It can be used to view batches with multiple program levels, but should not be used to edit such batches; any inserted or duplicated records will have the same program level as the record above.
- Mode M can be used to view regular data entry batches, and, with caution, to edit them. The LiFE-Works editor is completely independent of the formats defined for the batch being edited. Mode M does not execute any format code or validate data in any way; therefore, the normal safeguards against entering invalid data do not operate in mode M. (Mode M does conform to the record size defined for each format.)

- Mode M lets you establish and use tab settings in each file. When you open a batch using mode M, LiFE-Works checks for a tab file associated with your user ID. If you have a tab file, the settings in that file become the tab settings for the file you are editing.

If you do not have a tab file, LiFE-Works creates one for you and sets a tab stop on every fifth column, up to a maximum of 30 tab stops. You can alter these settings as necessary in the file you are editing. If you choose, you can write the altered settings into your tab file for future use.

- Mode M stores changed records in memory until you write them to disk. It's possible to run out of memory if you are changing a large number of records. It's also possible that you could lose your changes if the system fails. To avoid these problems, press CORR at regular intervals to write your changes to disk.

Table C-1. Mode M Function Keys

Key	Function
LEFT ARROW	Moves the cursor left one column.
RIGHT ARROW	Moves the cursor right one column.
UP ARROW	Moves the cursor up one line.
DOWN ARROW	Moves the cursor down one line.
RETURN	Moves the cursor to the start of the next record (cursor return).
HOME	Moves the cursor to the start of the current record.
ONLINE	Moves the cursor to the end of the text in the current record.
TAB	Moves the cursor one tab stop to the right.
SHIFT TAB	Moves the cursor one tab stop to the left.
INSERT	Inserts one blank. The rightmost character of the record is discarded.
DELETE	Deletes one character. The rightmost character of the record is set to a blank.
ILLEG	Copies the character immediately above the cursor into the current record.
VALID	Copies the character immediately below the cursor into the current record.
COPY ABOVE	Copies the remainder of the record above into the current record (all characters at or to the right of the cursor).
COPY BELOW	Copies the remainder of the record below into the current record (all characters at or to the right of the cursor).
ERASE EOF	Blanks the remainder of the current record (all characters at or to the right of the cursor).
ACCEPT or RECORD INSERT	Inserts a new, blank record after the current record.



Table C-1. Mode M Function Keys (Continued)

Key	Function
RECORD DELETE DUP	Delete
SCROLL	

THE MODE M SEARCH FUNCTION

Mode M's search function lets you search for a specified character string or line (record) in a file. Table C-2 summarizes the keys used in mode M searches.

Table C-2. Function Keys for Mode M Searches

Key	Function
SEARCH FWD	Searches forward (to the right and down) for a specified character string.
SEARCH BACK	Searches backward (to the left and up) for a specified character string.
FIND	Searches for a specific record number.
RETURN or DUP	Activates a search after you have entered a character string or record number.
EXIT	Cancels a search before it starts (after you have entered a search string but before you have pressed RETURN or DUP).
QUIT	Stops a search in progress.

L

SET

CLEAR

RESET

ATTN

CASE

To search for a character string:

- 1 Access the file in mode M.
- 2 Press SEARCH FWD to search forward, or SEARCH BACK to search backward.
- 3 Enter the search information. The search string can be up to 50 characters long and can contain any characters, including the operating system metacharacters ("wild-card" characters). Table C-3 lists the metacharacters you can use in searches.

For more information on the operating system metacharacters, see the System V/68 User's Reference Manual, Volume 1.

- 4 Press RETURN or DUP to start the search. (The search starts automatically if you type past the end of the Enter search information: prompt.) When LiFE-Works finds the specified string, it stops the search and places the cursor at the beginning of the string.
- 5 To continue searching for the same string, press SEARCH FWD or SEARCH BACK twice. LiFE-Works remembers the search string until you enter another search string or until you exit from LiFE-Works.

To search for a specific line by record number:

- 1 Access the batch in mode M.
- 2 Press the FIND key.
- 3 Enter the record number.
- 4 Press RETURN or DUP to start the search. (The search starts automatically if you type past the end of the Enter record number: prompt.) When LiFE-Works finds the record, it stops the search and places the cursor at the beginning of the record.

Table C-3. Operating System Metacharacters Used in Mode M Searches

Character	Function in Search String
? (question mark)	Searches for any single character. For example, <u>R??</u> searches for any three-letter word that starts with <u>R</u> .
* (asterisk)	Searches for any string of characters, including a null string. For example, <u>B*</u> searches for any word that starts with <u>B</u> .
[] (brackets)	Searches for any one of the characters enclosed in brackets. For example, <u>[123]</u> searches for any of the characters 1, 2, or 3.
- (minus sign)	When enclosed in brackets, indicates a range. For example, <u>[A-Z]</u> searches for any uppercase letter.
! (exclamation point)	Enclosed in brackets and in the first column position, signifies NOT (reverses the search criteria). For example, <u>[!A-Z]</u> finds all characters except uppercase letters.







## Index

In cases where commands or format elements begin with a special symbol (\$FORMAT, #BATCH, and so on), the symbol is ignored and the entry appears in its normal alphabetical sequence.

- \ (backslash) continuation character
  - In \$DEFJOB command, 1-10
  - In \$FORMAT command, 1-5
- \ (backslash) validation/generation descriptor, 5-41
- \$ EDIT picture symbol, 7-18, 7-19
- \* EDIT picture symbols, 7-17, 7-21
- + (plus) EDIT picture symbol, 7-18, 7-20
- , (comma) EDIT picture symbol, 7-18, 7-20
- (minus) attribute inhibitor, 4-10
- (minus) EDIT picture symbol, 7-18, 7-20
- . (decimal point) EDIT picture symbol, 7-19, 7-20
- () EDIT picture symbols, 7-18, 7-21
- /\* and \*/ comment delimiters, 7-12
  - In symbolic referencing, 11-4
- @r,c command, 6-17
- @w command, 6-18
- 0 EDIT picture symbol, 7-19, 7-21
- 9 EDIT picture symbol, 7-17, 7-19
- 3270/3274 communications, 12-1 through 12-10
- A field modifier, 3-3, 3-4, 3-5
- A field type, 2-3
- a field type, 2-4
- A validation/generation descriptor, 5-7
- \$ACCUMS subcommand of \$DEFJOB command, 1-10
- Accumulator 0, 5-7, 5-38, 7-42, 8-3
- Accumulator validation/generation descriptor, 5-7
- Accumulators
  - (See also accumulator 0)
  - Adding to, 3-6
  - Clearing (filling with zeros), 7-53
  - Double-precision, 1-10
  - Maintaining batch-balance totals in, 3-24
  - Setting values into, 7-39, 7-40
  - Single precision, 1-10
  - Specifying in \$DEFJOB command, 1-10
  - Subtracting from, 3-24
  - Validating against or generating from, 5-7
- Alphabetic check digits, B-5
- Alphabetic date system keyword, 5-12
- Alphabetic field type, 2-3
- Alphabetic field type with lowercase feature, 2-4
- Arithmetic expressions, 8-3, 8-4
  - In validation or generation, 5-5, 5-6, 5-7

## Index

- Arithmetic operators, 8-3, 8-4
- ASD command, 7-4
- ASD# command, 7-5
- Assigning symbols, 11-2, 11-3, 11-4, 11-7
  - In the SCREEN-SECTION, 11-5, 11-6
- ATR command, 4-9
- ATTN attention ID code, 12-3
- Attribute command, 4-9
- Attribute modifiers
  - General information, 4-1, 4-2, 4-3
  - Modifying the appearance of a given element, 4-4
  - Modifying the default display characteristic, 4-6
- Attribute spaces, 4-2, 4-3, 12-6
  - Specifying for a given format, 1-4
  - Viewing, 4-3, 12-3
- Attributes
  - (See also attribute modifiers, attribute spaces)
  - General information, 4-1, 4-2, 4-3
  - Inhibiting, 4-10
  - Moving, 4-8
  - Significance in interactive communications, 12-6
- \$ATTRSPACE YES/NO subcommand of \$FORMAT command, 1-4, 12-6
- Audible alarm command, 7-6
- Auto skip duplicate command, 7-4
- Auto skip duplicate off command, 7-5
- Automatic duplicate modifier, 3-10, 3-11
  - In formats for interactive communications, 12-3
- Auxiliary-duplicate modifier, 3-3, 3-4, 3-5
  
- B (blank) EDIT picture symbol, 7-18, 7-20
- B attribute modifier, 4-1, 4-3
- B field modifier, 3-6, 3-7
- B validation/generation descriptor, 5-8
- BACK command, 10-17 to 10-18
- Background color attribute modifier, 4-1, 4-3
- Balance-total modifier, 3-6, 3-7
- Base number for a check digit, B-1
- #BATCH system keyword, 5-9
- Batch balance total, maintaining, 3-24
- Batch balancing, use of Z command in, 7-53
- Batch identifier system keyword, 5-9
- BEEP command, 7-6
- BLANKB command, 7-7
- Blank to column command, 6-5
- Blank to row and column command, 6-4
- Blank work buffer command, 7-7
- Bn attribute modifier, 4-1, 4-3
- BRANCH command, 7-9, 7-10
- Branching statements, 8-8 through 8-28
- Br,c command, 6-4
- BROWSE command, 10-17, 10-19, 10-20
- Browse through index set records command, 10-17, 10-19, 10-20

- BUFFER conditional indicator, 8-17
- BUFFER keyword, 11-2, 11-4
- BUFFER-SECTION keyword, 11-2, 11-3, 11-4
- Buffer, work
  - Allocating, 7-27
  - Blanking, 7-7
  - Releasing, 7-37
  - Validating against or generating from, 5-8
- Buffer, work, system keyword, 5-8
- Bw command, 6-5
- C field modifier, 3-8, 3-9
- CALL command, 7-11
- Calling a subroutine, 7-11
- Character color attribute modifier, 4-1, 4-3
- Character strings, validating against or generating from, 5-39, 5-40, 5-41
- Characters, inhibiting screen display of, 7-15
- Check digit calculation
  - Alphabetic check digits, B-5
  - C command, 7-10
  - C modifier, 3-8, 3-9
  - Modulus 7, B-2
  - Modulus 10, B-3
  - Modulus 11, B-4
- Check digit modifier, 3-8
- CLEAR attention ID code, 12-3
- CLEARMDT format command, 12-9
- Clearing the screen during interactive communications, 12-3
- Cn attribute modifier, 4-1, 4-3
- Color attribute modifiers, 4-1, 4-3
- Commands, 7-1 through 7-54
  - (See also individual names)
  - Attribute-related, 4-6 through 4-10
  - For creating a format, 1-4, 1-5, 1-6, 1-7
  - Conditional, and conditional indicators, 8-15 through 8-28
  - For defining a job, 1-10, 1-11, 1-12
  - Index-set, 10-1 through 10-32
  - Screen-formatting, 6-1 through 6-18
- COMMENT command, 7-12
  - In symbolic referencing, 11-4
- Comment command, 7-12
  - In symbolic referencing, 11-4
- Communications with a host, 12-1 through 12-10
- Compiling a format, 1-8, 1-9
- Compiler directives, 7-1
- Conditional commands and conditional indicators, 8-15 through 8-28
  - (See also individual names)
- Conditional statements, 8-8 through 8-28
- Conditional verify command, 8-21
- Conditional verify indicator, 8-18
- #CONST system keyword, 5-10



## Index

- Continuation character (\)
  - In \$DEFJOB command, 1-11
  - In \$FORMAT command, 1-4
  - In \$INDSET command, 10-4
- Copying a format, 1-13
- CORR command, 7-13
- Correction command, 7-13
- CR EDIT picture symbol, 7-18, 7-20
- D (data) modifier
  - With F descriptor, 5-15, 5-16
  - With LF descriptor, 5-18, 5-19
  - With RS descriptor, 5-24, 5-25
  - With S descriptor, 5-26, 5-27
- D field modifier, 3-10, 3-11
  - In formats for interactive communications, 12-3, 12-4
- Data definition
  - Copying, 13-12
  - Creating, 13-4 through 13-9
  - Deleting, 13-11
  - Modifying, 13-10
- Data dictionary
  - Data Dictionary Generator Menu, 13-3, 13-4, 13-10, 13-11
  - Starting the Data Dictionary Generator, 13-3
- Data, ensuring that characters are treated as
  - In a field, 5-15, 5-16
  - In a local field, 5-18, 5-19
  - In a relative screen area, 5-25, 5-26
  - In a screen area, 5-27
- #DATE system keyword, 5-11
- #DATEL system keyword, 5-12
- #DAY system keyword, 5-14
- Day system keyword, 5-14
- DB EDIT picture symbol, 7-18, 7-20
- DECIMAL POINT IS COMMA directive, 7-19
- DECIMAL POINT IS PERIOD directive, 7-19
- DECLARATIONS area, 11-2, 11-3, 11-4
  - Comments in, 11-4
  - Multiple symbol assignments in, 11-7
  - SCREEN-SECTION of, 11-5, 11-6
- Default display characteristic, changing, 4-6
- Defining a job, 1-10, 1-11, 1-12
- Defining a subroutine, 7-46, 7-47
- \$DEFJOB command, 1-10, 1-11, 1-12
- \$DELETE command, 1-15, 1-16
- Delete index set record command, 10-16, 10-21
- Deleting a system element (format, index set, and so on), 1-15, 1-16
- DELIXR command, 10-16, 10-21
- Descriptors, validation/generation, 5-1 through 5-42
  - (See also individual names)
- Directives, compiler, 7-1
- Disable EXIT key command, 7-33
- DISPLAY command, 7-14

DISPLAY# command, 7-15  
 Display-only modifier, 3-19  
 \$DOCHDR subcommand of \$FORMAT, 1-4  
 Document header, assigning a format as a, 1-4  
 DOCVER conditional indicator, 8-18  
 \$DOUBLE subcommand of \$FORMAT command, 1-4  
 Duplicate key field values, 10-3  
   And INSIXR command, 10-23  
 DUPS parameter of \$INDSET command, 10-3  
 E field modifier, 3-12  
 EDIT command, 7-16 through 7-21  
 Edit numeric field command, 7-16 through 7-21  
 Enable EXIT key command, 7-31  
 END command, 7-46  
 Ending a session, 12-2  
 Ending a subroutine, 7-22  
 END-OF-DECLARATIONS keyword, 11-2, 11-3, 11-4  
 ENDSUB command, 7-22  
 ENTER attention ID code, 12-3  
 Entering format code, 1-4, 1-5, 1-6, 1-7  
 Er,c command, 6-6, 6-7  
 Erase columns command, 6-8  
 Erase to row and column command, 6-6, 6-7  
 Erase unprotected columns command, 6-9, 6-10  
 Erase unprotected columns to row and column command, 6-9  
 Error conditional command, 7-20  
 Error conditional indicator, 7-21  
 Error log, 12-10  
 EUAr,c command, 6-9, 12-9  
 EUAw command, 6-10, 12-9  
 European dates, 5-11, 5-12  
 European-style decimal, 7-19  
 Ew command, 6-8  
 Exclusive access conditional indicator, 10-30  
 Exclusive access key field or next highest key field validation  
   descriptor, 10-6, 10-8, 10-10  
 Exclusive access key field validation descriptor, 10-6, 10-8  
 EXCLUSIVE conditional indicator, 10-30  
 EXEC command, 7-23, 7-24  
 EXIT key, enabling/disabling, 7-31, 7-32  
 Exit format conditional command, 8-19  
 EXITF conditional command, 8-18  
 Expressions, arithmetic, 8-3, 8-4  
   In validation or generation, 5-4, 5-5, 5-6  
  
 F attribute modifier, 4-1  
 F command, 7-25  
 F field modifier, 3-13  
 F validation/generation descriptor, 5-15, 5-16  
   In subroutines, 7-47  
 FDR command, 7-26

## Index

- Field definition, 13-1
  - Creating, 13-5, 13-6, 13-7, 13-8
  - Deleting, 13-11
  - Modifying, 13-10
- Field modifiers, 3-1 through 3-24
  - (See also individual names)
- Field numbers, 2-2
  - In a subroutine, 7-47
- Field reference validation/generation descriptor, 5-15, 5-16
- Field scan comparisons, use of in validation/generation, 5-7
- Field types, 2-1 through 2-17
  - (See also individual names and Fields, entry)
- Fields, entry, 2-1 through 2-17
  - Attribute modifiers and, 4-1 through 4-3
  - Editing numeric, 7-16 through 7-21
  - Field numbers and, 2-2, 7-47
  - Setting values into, 7-38, 7-39, 7-40
  - Validating against or generating from, 5-15, 5-16
    - In a subroutine, 5-18, 5-19, 7-47
- FILE=fn keyword, 11-2
- FILLER keyword, 11-3, 11-4
  - In SCREEN-SECTION, 11-5
- Force disk record command, 7-26
- \$FORMAT command, 1-4, 1-5, 1-6, 1-7, 12-6
- \$FORMAT subcommand of \$DEFJOB command, 1-10, 1-11, 1-12
- Format Generator
  - Compiling the Code, 14-12
  - Creating Data Fields, 14-6
  - Creating Screen Prompts, 14-6
  - Defining Data Fields, 14-7
  - Designing the Input Screen, 14-4
  - Function Keys, 14-5, 14-6
  - Starting, 14-2
- Formats
  - Assigning to a job, 1-10, 1-11, 1-12
  - Compiling, 1-8, 1-9
  - Copying, 1-13, 1-14
  - Deleting, 1-15, 1-16
  - Entering code for, 1-4, 1-5, 1-6, 1-7
  - Identifier, 1-4, 1-8
  - Source code and object code, defined, 1-8
  - Subroutines, 7-47, 7-48, 7-49
- G field modifier, 3-14, 3-15, 3-16
- G field type, 2-6
- g field type, 2-7
- General field type, 2-6
- General field type with lowercase feature, 2-7
- Generate check digit command, 7-10
- Generate modifier, 3-14, 3-15, 3-16
- Generate text command, 6-15, 6-16
- Generation, 5-1 through 5-42
  - From an index set, 10-6, 10-7, 10-8



- Generation/validation descriptors, 5-1 through 5-42
  - Index set, 10-9 through 10-16
  - (See also individual names)
- Get work buffer command, 7-27
- GETBUF command, 7-27
- H attribute modifier, 4-1, 4-3
- H field type, 2-9
- Hexadecimal field type, 2-9
- Hexadecimal string constant validation/generation descriptor, 5-42
- I field type, 2-10
- IF...THEN...ELSE statements, 8-9 through 8-12
- INCLUDE compiler directive, 7-28
- Include job and batch directive, 7-28
- Index set commands, 10-16
- Index set record field validation/generation descriptor, 10-8 through 10-15
- Index set select conditional indicator, 10-31, 10-32
- Index sets
  - Assigning symbols to fields in, 11-2, 11-3, 11-4
  - Common errors in creating, 10-4
  - Conditional indicators, 10-31
  - Creating, 10-3
  - Definition of, 10-1, 10-2
  - Deleting, 1-15, 1-16
  - General information, 10-1, 10-2
  - Key fields, 10-1 through 10-4
  - Keynames, 10-3
  - SET statements and, 10-14, 10-15
  - Setting values in fields from other index sets, 10-14
  - Releasing all access to, 10-27
  - Releasing exclusive access to, 10-26
  - Types of access, 10-6
  - Using more than once in the same format, 10-13
  - Validation/generation descriptors, 10-5, 10-6, 10-7
- Index-generate modifier, 3-17
- INDEX-id keyword, 11-2
- INDEX-SECTION keyword, 11-2, 11-3, 11-4
- \$INDSET command, 10-3, 10-4
- INFORMIX, 13-4
- Inhibit character display command, 7-15
- Inhibiting attribute generation, 4-10
- Insert index set record command, 10-17, 10-22, 10-23
- Inserting blanks into a record, 3-21
- INSIXR command, 10-16, 10-22, 10-23
- Integer field type, 2-10
- INTEN command, 4-6, 4-7
- INTEN compiler directive, 4-6, 4-7
- Interactive communications with a host, 12-1 through 12-10
- ISAM flag, 10-4
- \$IXBATCH parameter, 10-3, 10-4
- Ix field modifier, 3-17



## Index

- J field modifier, 3-18
- #JOB system keyword, 5-17
- Job, defining a, 1-10, 1-11, 1-12
- Jobname system keyword, 5-17
- Justify modifier, 3-18
  
- K validation descriptor, 10-8
- Key field, index set
  - Defining in the \$INDSET command, 10-3, 10-4
  - Defining in a data definition, 13-7, 13-8, 13-9
- Key verify conditional indicator, 8-20
- \$KEYFLD subcommand of \$INDSET command, 10-3, 10-4
- KEYVER conditional indicator, 8-20
- Keywords, system, 5-1
  - Shortening the value of, 5-4
- KN validation descriptor, 10-10
- KNS validation descriptor, 10-11
- KS validation descriptor, 10-9
- KVDOC conditional command, 8-21
  
- L attribute modifier, 4-1, 4-3
- L field type, 2-11, 2-12
- Left-justified-with-no-embedded-blanks validation descriptor, 5-20
- Left-zero-fill field type, 2-11, 2-12
- Left-zero-fill field type with implied decimal, 2-13, 2-14
- Levels in DECLARATIONS area, 11-2 through 11-6
- LF validation/generation descriptor, 5-18, 5-19, 7-48, 7-49
- ?LJNIB validation descriptor, 5-20
- Local field validation/generation descriptor, 5-18, 5-19, 7-48, 7-49
- Logic and arithmetic, 8-1 through 8-28
- Logical AND (&), 8-5, 8-7
  - In logical tests used as alternative to value set validation, 5-36
  - In validation and generation, 5-5
- Logical OR (|), 8-5, 8-6
  - In logical tests used as alternative to value set validation, 5-36
  - In validation and generation, 5-5
  - Use of | symbol in string constant validation, 5-39
- LOW command, 7-29
- LOW# command, 7-30
- Lowercase feature on command, 7-29
- Lowercase feature off command, 7-30
- Lowercase, using in format code, 1-5
- Lw.d field type, 2-13, 2-14
  
- Mainframe configuration for interactive communications, 12-5
- MARK conditional command, 8-22
- MARKED conditional indicator, 8-23
- \$MAXCOL subcommand of \$FORMAT command, 1-4
- \$MAXROW subcommand of \$FORMAT command, 1-4
- \$MAXSIZ (\$MAXSIZE) subcommand of \$DEFJOB command, 1-10
- MDT bit, 12-7, 12-8
- MODE command, 7-31

Mode I (vi), 1-7  
 Mode M, 1-7, C-1 through C-8  
 MODE# command, 7-32  
 Modified data tag (MDT) bit, 12-7, 12-8  
 Modulus 7, B-2  
 Modulus 10, B-3  
 Modulus 11, B-4  
 Modulus calculation, B-1  
 MSFP, 14-1  
 Multiple symbol assignments, 11-7  
 Multiple-key index sets, 10-2  
     And INSIXR command, 10-23  
 Multirecord editor, 1-7, C-1 through C-8  
 Multistation File Processing (MSFP), 14-1  
 Must-enter modifier, 3-12  
 Must-fill modifier, 3-13  
 Must-release modifier, 3-20  
  
 N field type, 2-15  
 NDR command, 7-33  
 ?NEG validation descriptor, 5-21  
 Negative numbers in left-zero-fill fields, 2-11, 2-12, 2-13, 2-14  
 Negative overpunch characters,  
     Checking for, 5-21  
     In left-zero-fill fields, 2-11, 2-12, 2-13, 2-14  
 Negative-overpunch character validation descriptor, 5-21  
 Nesting conditional statements, 8-12  
 New record conditional indicator, 8-24  
 NEWREC conditional indicator, 8-24  
 NEXT command, 10-16, 10-24, 10-25  
 No backspacing command, 7-34  
 No disk record command, 7-33  
 No record/document up command, 7-35  
 NOBACK command, 7-35  
 Nonmatching lengths, 7-41 through 7-43  
 NOUP command, 7-35  
 NULL conditional command, 8-25  
 Numeric field type, 2-15  
 Numeric date system keyword, 5-11  
  
 Object code  
     Definition of, 1-8  
     Deleting, 1-15, 1-16  
     Copying, 1-13, 1-14  
 OfficeLAN node name, 13-5, 13-12  
 ONLINE command, 12-1, 12-2  
 ONLINE# command, 12-2  
 ONLINE conditional indicator, 12-3  
 Opening a record for correction under format control, 7-13  
 #OPER system keyword, 5-22  
 Operating system user ID system keyword, 5-34  
 Operator ID system keyword, 5-22  
 OUTBAL conditional indicator, 8-26  
 Out-of-balance conditional indicator, 8-26

## Index

### P (prompt) modifier

- With F descriptor, 5-15, 5-16
- With LF descriptor, 5-18, 5-19
- With RS descriptor, 5-24, 5-25
- With S descriptor, 5-26, 5-27

### P command, 6-11, 6-12

### PA1 through PA3 attention ID codes, 12-3

### Pass two conditional indicator, 8-27

### \$PASSW (\$PASSWORD) subcommand of \$DEFJOB command, 1-10, 1-11

### Password, assigning to a job, 1-10, 1-11, 1-12

### PASTWO conditional indicator, 8-27

### \$PAUSE command, 1-6

### PF1 through PF12 attention ID codes, 12-3

### Picture symbols for numeric field editing, 7-18 through 7-22

### Pr,c command, 6-13

### Primary key field, 10-3, 10-4

### PRINTS command, 7-36

### Program level command, 7-25

### Prompt display command, 6-11, 6-12

### Prompt display at row and column command, 6-13, 6-14

### Prompts

#### Displaying on the screen, 6-11 through 6-14

#### Ensuring that generated characters are treated as

- In a field, 5-15, 5-16
- In a local field, 5-18, 5-19
- In a relative screen area, 5-24, 5-25
- In a screen area, 5-26, 5-27

### Program access operations in interactive communications, 12-3

### Program function operations in interactive communications, 12-3

### Pw command, 6-11, 6-12

### Q field modifier, 3-19

### Quotation marks in format code, 1-6

### R attribute modifier, 4-1, 4-3

### R command, 10-16, 10-26

### R field modifier, 3-20

### RA command, 10-16, 10-27

### Range check validation using < and > relational operators, 5-38, 5-39, 5-40

### #RECNO system keyword, 5-23

### RECORD-SECTION keyword, 11-2, 11-3, 11-4

### Record number system keyword, 5-23

### REDEFINES keyword, 11-7

### Redefining symbol assignments, 11-7

### Referencing, symbolic, 11-1 through 11-8

### RELBUF command, 7-37

### Relational operators, 8-1

#### In validation, 5-4, 5-5, 5-38

### Relative screen area validation/generation descriptor, 5-24, 5-25, 7-48

### Release all access command, 10-16, 10-27

### Release exclusive access command, 10-16, 10-26

### Release work buffer command, 7-37



- RESETMDT format command, 12-9
- Resume character display command, 7-15
- Rfn validation/generation descriptor, 5-2
- RS validation/generation descriptor, 5-24, 5-25, 7-47
- S field modifier, 3-21
- S validation/generation descriptor, 5-26, 5-27
- SA validation/generation descriptor, 5-28, 5-29
- Scan-for-any validation descriptor, 5-28, 5-29
- Scan-for-none validation descriptor, 5-30, 5-31
- Screen area
  - In a subroutine, 7-47
  - Setting a value in a, 7-38, 7-39, 7-40
- Screen area validation/generation descriptor, 5-26, 5-27
- Screen formatting commands, 6-1 through 6-18
  - (See also individual names)
- SCREEN keyword, 11-1, 11-2, 11-3, 11-4, 11-5
- Screen print format command, 7-36
- \$SCREEN subcommand of \$FORMAT command, 1-4
- SCREEN=SECTION keyword, 11-2, 11-3, 11-4, 11-5
- Searching for a character string or record number (mode M), C-6, C-7, C-8
- Select next index set record command, 10-16, 10-24, 10-25
- Select previous index set record command, 10-16, 10-17, 10-18
- SELREC conditional indicator, 10-29, 10-31
- SEND command, 12-3, 12-4
- Sending data to a host, 12-1 through 12-10
- Sessions, 12-1, 12-2
- SET command, 7-38, 7-39, 7-40
  - Nonmatching lengths, 7-41, 7-42, 7-43
  - Using with index sets, 10-14, 10-15
- SETMDT format command, 12-9
- SHARED conditional indicator, 10-29, 10-32
- Shared access conditional indicator, 10-29, 10-32
- Shared access key field or next highest key field validation descriptor, 10-6, 10-11
- Shared access key field validation descriptor, 10-6, 10-9
- Sharing symbol assignments, 11-8
- \$SIZE subcommand of \$DEFJOB command, 1-10
- \$SIZE subcommand of \$FORMAT command, 1-4
- Sight verify command, 7-52
- Sight verify, flagging a record for display in, 7-52
- Skip modifier, 3-21
- SLEEP command, 7-44, 7-45
- SN validation/generation descriptor, 5-30, 5-31
- Source code, definition of, 1-8
- SPLIT command, 8-13, 8-14
  - And value set validation, 9-3
- Starting a 3270 session, 12-1
- Stopping a 3270 session, 12-1
- String constant validation/generation descriptor, 5-38, 5-39, 5-40
- \$SUB subcommand of \$FORMAT command, 1-5, 7-47
- SUBROUTINE command, 7-46, 7-47



## Index

SUPERM command, 7-48, 7-49  
SUPERS command, 7-48, 7-49  
Supervisory format commands, 7-48, 7-49  
Suspend format execution command, 7-44  
SWITCH command, 10-16, 10-28  
Switch key command, 10-16, 10-28  
\$SYM subcommand of \$FORMAT command, 1-5  
.Symbol. validation/generation descriptor, 5-37  
Symbols and symbolic referencing, 11-1 through 11-8  
    .Symbol. validation/generation descriptor, 5-37  
Symbols, nontraditional, used by LIfE-Works, 1-6  
SYSREQ attention ID code, 12-3  
System constant system keyword, 5-10  
System keywords, 5-1  
    (See also individual names)  
    Shortening the value of, 5-4  
System-generated characters, screen display of, 7-16, 7-17  
  
T command, 6-15, 6-16  
Tab to column command, 6-18  
Tab to row and column command, 6-17  
Terminal number system keyword, 5-32  
#TERMNO system keyword, 5-32  
Test request, sending during interactive communications, 12-3  
Text command, 6-15, 6-16  
"Text" validation/generation descriptor, 5-38, 5-39, 5-40  
Time system keyword, 5-33  
#TIMES system keyword, 5-33  
Trail verify, 8-15  
TREQ attention ID code, 12-3  
Transferring control from one format to another, 7-8, 7-9  
Tw command, 6-15  
  
U attribute modifier, 4-1  
Uppercase, using in format code, 1-5  
User ID system keyword, 5-34  
#USERID system keyword, 5-34  
  
V command, 7-50  
V validation/generation descriptor, 5-35, 5-36  
V# command, 7-51  
VALID command, 7-52  
Validation/generation descriptors, 5-1 through 5-42  
    (See also individual names)  
Validation, 5-1 through 5-42  
Value sets, 9-1 through 9-3  
    Assigning to a job, 1-10  
    Generation from, 3-17, 9-1  
    Validation against, 5-36, 5-37, 9-3  
Value set validation descriptor, 5-36, 5-37, 9-3  
\$VALUES subcommand of \$DEFJOB command, 1-10  
Verify off command (V#), 7-51

- Verify on command (V), 7-50
- Verify reconstruct conditional indicator, 8-28
- VRECON conditional indicator, 8-28
- W field type, 2-16, 2-17
- WAITW command, 12-3, 12-4, 12-5
- Word processing field type, 2-16, 2-17
- Work buffer
  - Allocating, 7-27
  - Blanking, 7-7
  - Releasing, 7-37
  - Validating against or generating from, 5-8
- Work buffer validation/generation descriptor, 5-8
- X validation/generation descriptor, 10-12, 10-13
  - In a SET command, 10-14, 10-15
- X field modifier, 3-22, 3-23
- Z command, 7-53
- Z EDIT picture symbol, 7-17, 7-20
- Z field modifier, 3-24
- Zero accumulator command, 7-53
- Zero-balance modifier, 3-24







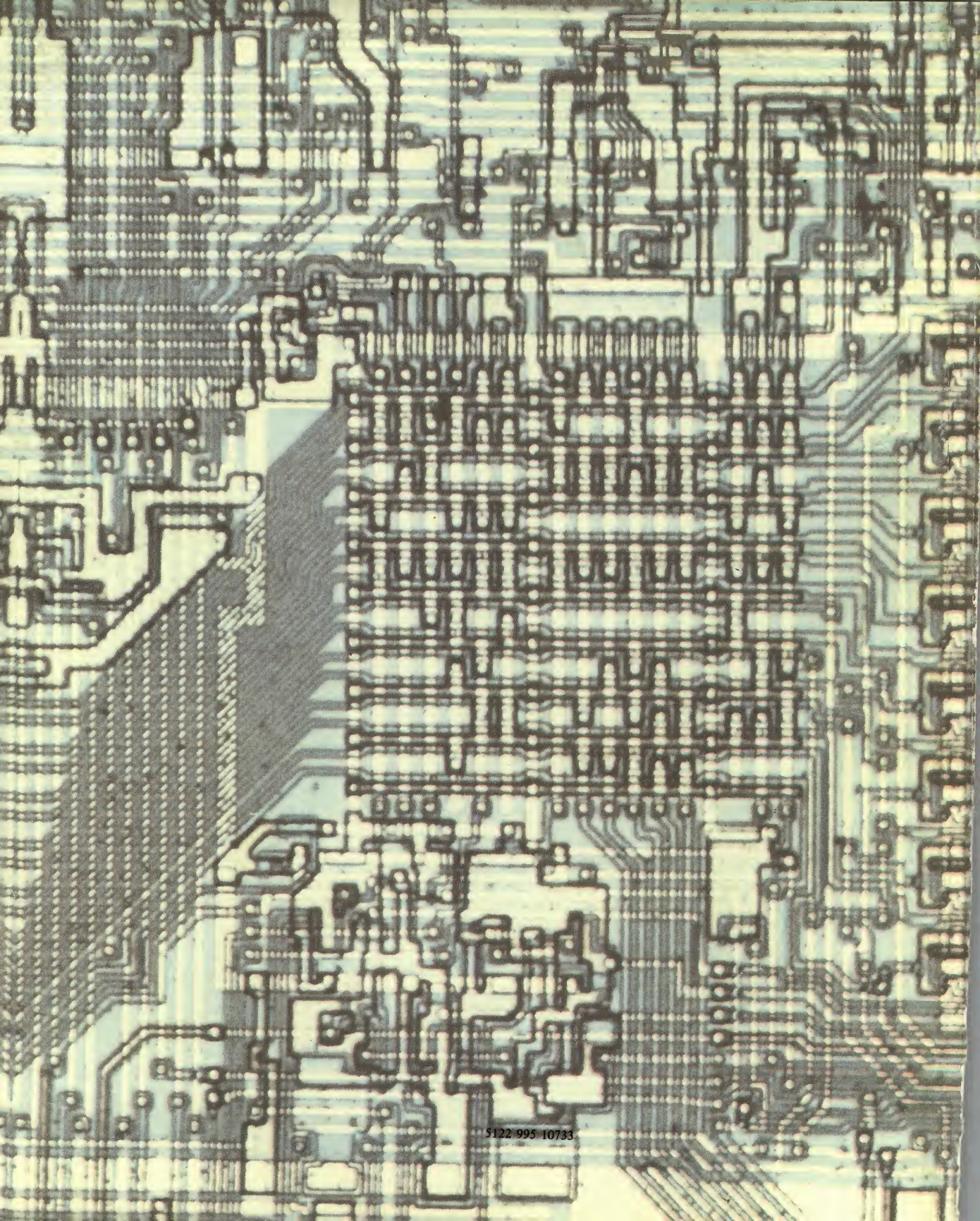












5122 995 10733